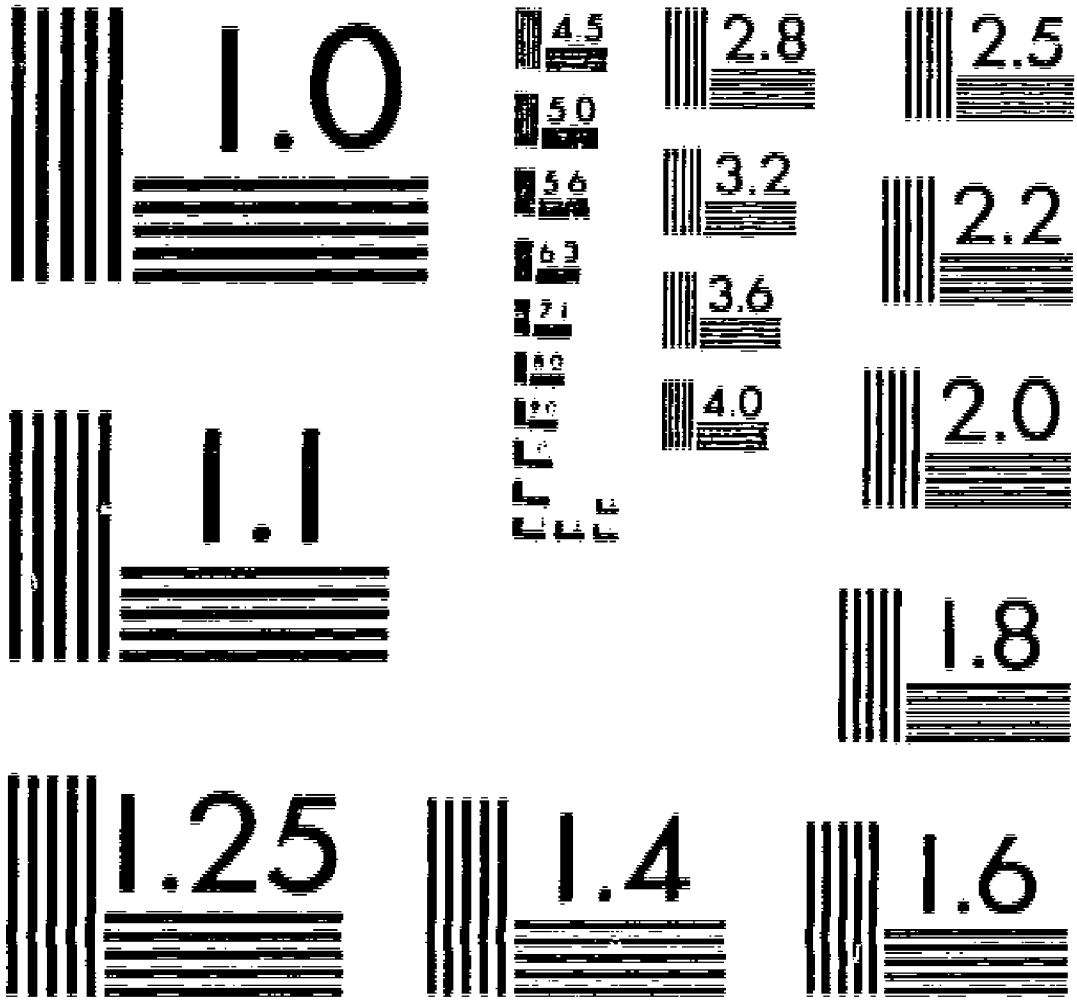


MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

DOCUMENT RESUME

ED 060 920 52 LI 003 611

AUTHOR Smith, Stephen F.; Harrelson, William
TITLE TMS: A Terminal Monitor System for Information Processing. Final Report.
INSTITUTION California Univ., Berkeley. Inst. of Library Research.
SPONS AGENCY Office of Education (DHEW), Washington, D.C. Bureau of Research.
BUREAU NO BR-7-1085
PUB DATE Sep 71
GRANT OEG-1-7-071085-4286
NOTE 131p.; (0 References)

EDRS PRICE MF-$0.65 HC-$6.58
DESCRIPTORS *Automation; Computer Programs; Data Bases; Electronic Data Processing; *Information Processing; *Information Retrieval; *Library Education; *Library Science; Manuals; On Line Systems; Research
IDENTIFIERS *University of California Berkeley

ABSTRACT

The results of the second 18 months (December 15, 1968 - June 30, 1970) of effort toward developing an Information Processing Laboratory for research and education in library science is reported in six volumes. This volume contains two parts. Part I includes: a user's guide - a guide to writing programs to TMS (Terminal Moniter System) for information processing. Part II is a system programmer's guide to the internal structure of TMS itself. The information presented in Part II is of critical importance to anyone interested in expanding or modifying the existing capabilities of TMS. (Other volumes of this report are available as LI 003607 through 003610). (Author/NH)

FINAL REPORT
Project No. 7-1085
Grant No. OEG-1-7-071085-4286

TMS: A TERMINAL MONITOR SYSTEM FOR INFORMATION PROCESSING

By

Stephen F. Smith
William Harrelson

Institute of Library Research
University of California
Berkeley, California 94720

September 1971

The research reported herein was performed pursuant to a grant with the Office of Education, U.S. Department of Health, Education, and Welfare. Contractors undertaking such projects under Government sponsorship are encouraged to express freely their professional judgment in the conduct of the project. Points of view or opinions stated do not, therefore, necessarily represent official Office of Education position or policy.

U.S. DEPARTMENT OF
HEALTH, EDUCATION, AND WELFARE
Office of Education (DHEW), Washington
Bureau of Research

TABLE OF CONTENTS

PART I: A GUIDE TO WRITING PROGRAMS FOR TMS

TABLE OF CONTENTS (Cont.)

APPENDICES

PART II: A PROGRAM LOGIC MANUAL FOR THE TERMINAL MONITOR SYSTEM

TABLE OF CONTENTS (Cont.)

APPENDICES

LIST OF FIGURES

FOREWORD

This report contains the results of the second 18 months (December 15, 1968 - June 30, 1970) of effort toward developing an Information Processing Laboratory for research and education in library science. The work was supported by a grant (OEG-1-7-071085-4286) from the Bureau of Research of the Office of Education, U.S. Department of Health, Education, and Welfare and also by the University of California. The principal investigator was M.E. Maron, Professor of Librarianship.

This report is being issued as six separate volumes by the Institute of Library Research, University of California, Berkeley. They are:

- Maron, M.E. and Don Sherman, et al. An Information Processing Laboratory for Education and Research in Library Science: Phase 2.

  Contents--Introduction and Overview; Problems of Library Science; Facility Development; Operational Experience.

- Mignon, Edmond and Irene L. Travis. LABSEARCH: ILR Associative Search System Terminal Users' Manual.

  Contents--Basic Operating Instructions; Commands; Scoring Measures of Association; Subject Authority List.

- Meredith, Joseph C. Reference Search System (REFSEARCH) Users' Manual.

  Contents--Rationale and Description; Definitions; Index and Coding Key; Retrieval Procedures; Examples.

- Silver, Steven S. and Joseph C. Meredith. DISCUS Interactive System Users' Manual.

  Contents--Basic On-Line Interchange; DISCUS Operations; Programming in DISCUS; Concise DISCUS Specifications; System Author Mode; Exercises.

- Smith, Stephen F. and William Harrelson. TMS: A Terminal Monitor System for Information Processing.

  Contents--Part I: Users' Guide - A Guide to Writing Programs for TMS
  Part II: Internals Guide - A Program Logic Manual for the Terminal Monitor System

- Aiyer, Arjun K. The CIMARON System: Modular Programs for the Organization and Search of Large Files.

  Contents--Data Base Selection; Entering Search Requests; Search Results; Record Retrieval Controls; Data Base Generation.

Because of the joint support provided by the File Organization Project (OEG-1-7-071083-5068) for the development of DISCUS and of TMS, the volumes concerned with these programs are included as part of the final report for both projects. Also, the CIMARON System, whose development was supported by the File Organization Project, has been incorporated into the Laboratory operation and therefore, in order to provide a balanced view of the total facility obtained, that volume is included as part of this Laboratory project report. (See Shoffner, R.M., et al., The Organization and Search of Bibliographic Records in On-Line Computer Systems: Project Summary.)

ACKNOWLEDGMENTS

Many people - faculty, Institute staff, and students of the School of Librarianship - helped to create the existing Terminal Monitor System. Especially notable among the programmers who developed and tested several utility modules are Chakravarthi Ravi, Arjun Aiyer, and Rodney Randall. These people and others also were instrumental in pinpointing and correcting design weaknesses. Ralph Shoffner, Allan Humphrey, and Don Sherman were key figures in prescribing additional facilities necessary for the system's smooth operation.

Principal acknowledgements are due to both the School of Librarianship of the University of California and the Office of Education of the Department of Health, Education, and Welfare, for making this work possible.

In addition, we wish to thank and to commend the work of the Institute personnel who prepared these pages for publication: Ellen Drapkin, Carole Fender, Bettye Geer, Linda Herold, Jan Kumataka, and Rhozalyn Perkins.

PREFACE

The cathode ray tube terminals of the Information Processing Laboratory are linked to a remote IBM 360 computer. This computer is run under the IBM Operating System, which allows several different users to share simultaneously the resources of the machine. This sharing is accomplished by establishing independent "partitions" of main memory and allocating one partition to each active program. The Operating System maintains control over and provides services to the individual computer programs residing in the various partitions. This is on a one-to-one basis; i.e., the Operating System allows but one program to be active in a given partition at any given time.

Since this 360 is not dedicated to serving the Information Processing Laboratory, it is necessary that the entire network of remote terminals appear to the Operating System to be a single program residing in one partition. However, each remote terminal must be able to call forth and use individual programs independently of the simultaneous activity on other terminals. Thus, to serve the needs of the Information Processing Laboratory, there must be a means of running several independent programs simultaneously within a single partition that is under the control of the Operating System.

To meet this need, the Institute of Library Research has developed the Terminal Monitor System (TMS). This is a system program which provides the software interface between the Operating System and applications programs running on individual terminals. It provides terminal programs with access to the services offered by the Operating System, and it also serves to represent the entire network as a single program to the Operating System.

This two-part volume of the final report on the Information Processing Laboratory project describes the Terminal Monitor System. Part I, an application programmer's guide to TMS, tells how to use TMS facilities to write on-line application programs that are to be run on the Laboratory network. Part II is a system programmer's guide to the internal structure of TMS itself. The information presented in Part II is of critical importance to anyone interested in expanding or modifying the existing capabilities of TMS.

PART I

A GUIDE TO WRITING PROGRAMS FOR TMS

1. INTRODUCTION

This manual is intended for use by programmers writing application programs to be run under the Terminal Monitor System (TMS). As such, it gives a brief overview of system design and details the programming conventions to be employed when using this system, as well as the specifications for employing the system macro instructions. It also describes an elementary top-level terminal control language that allows the user at a terminal to identify himself (i.e., "log in") to the system, specify the problem program that he wishes to work with, and recover from program errors. A much greater level of detail regarding system design, the actual expansion of system macro instructions, and details on the system processing modules is contained in a companion publication, the TMS Internals Guide.

This manual assumes on the part of the problem programmer a moderate level of proficiency in writing Assembler Language programs to operate under OS/360. This should include basic Assembler Language programming and the use of the Supervisor Services and Data Management Services macro instructions. Specifically, this manual assumes a working knowledge of the contents of the following publications:

a. System 360 Principles of Operation, Form A22-6821

b. OS/360 Concepts and Facilities, Form C28-6535

c. OS/360 Assembler Language, Form C28-6514

d. OS/360 Assembler F Programmers Guide, Form C26-3756

e. OS/360 Job Control Language, Form C28-6539

f. OS/360 Supervisor and Data Management Services, Form C28-6646

g. OS/360 Supervisor and Data Management Macro Instructions, Form C28-6647

1.1 Criteria Used in Design

Several basic criteria have governed the de gn of TMS. Primary among these has been the need to fit the system into rather a small amount of main storage (compared to systems with similar capabilities). Of nearly equal importance has been the need to maintain complete compatibility with the IBM Operating System (OS/360) with minimal (hopefully none) alterations to OS code. The system has to be able to service the several users on the individual terminals in a quasi-simultaneous manner and give these users the options of running different application programs or sharing the same application program with little or no consideration as to what the other users of the system may be doing at that particular

time. As a corollary, the system has to remain as immune as possible to crashes by individual application programs and maintain service to the remaining terminals while attempting to recover from any problems encountered. Due to the past history of ILR equipment acquisitions and possible changes in the future, the system must be able to deal with different types of terminals including a mixture of keyboard and cathode-ray tube (CRT) terminals. Insofar as is possible, the potential effects on the application programmer of having to deal with different types of terminals should be minimized. Finally, an elementary form of top-level control language was found necessary to perform such housekeeping functions as the identification of authorized users and the loading of programs at their request.

The above criteria, in addition to consideration of the available manpower and resources, resulted in the following primary design decisions. First, the amount of monitor code in main storage at execution time had to be minimized; thus all setup and shutdown functions had to be separated out from this code and assembled as separate load modules to be in main storage only during setup and shutdown processes. The need for OS compatibility and maximum reliability required a system that would stand between the user and OS/360 and edit all requests by application programs to reduce or eliminate the possibility of system crashes in trying to serve these requests. It was decided that as much as possible of the burden of checking a user's service request for validity would be placed on the assembly phase of application program development by building extensive checking facilities into the various TMS macros and thus eliminating the need for execution time checking in many cases. Finally, space and manpower requirements dictated that while the system exhibit certain features commonly found only in so-called time-sharing systems or in OS/MVT the system itself would not be designed in this manner; thus, the system design achieves a level of complexity somewhere between the dedicated application teleprocessing system and the full time-sharing system.

These primary design decisions led in a fairly natural manner to certain secondary design decisions. Since on a small computer or in a small multi-programming partition the greatest problem likely to occur is running out of space, first priority had to be given to detecting the impending occurrences of this problem and avoiding them. Since the most uncontrollable situation with respect to core allocation generally arises with the execution of an OS OPEN macro instruction certain steps had to be taken to minimize the possible adverse effect of using this macro. One of the most important was in pre-loading all of the necessary access method subroutines into the TMS partition prior to starting execution of TMS. To minimize the amount of core used up in this manner it was decided to use only the basic access methods and to restrict availability to a subset of basic access methods services which would adequately serve the various application programmers. With the loading of access method subroutines under control, and the possibility of

automatic buffer allocation greatly reduced by limiting ourselves to basic access methods, most of the remaining core allocation problems became easily predictable and could be handled by doing conditional GETMAIN's. Further, to minimize the amount of code necessary to set up an OPEN macro instruction, the data sets were limited to direct access data sets already allocated and cataloged in the system catalog. Thus data set availability could be readily checked and most data set parameters would be obtained from the various data set control blocks (DSCB's).

To allow TMS to deal with many different types of terminals in a relatively uniform manner with minimal investments in special programming the Basic Teleprocessing Access Method (BTAM) was selected as the interface between TMS and the various terminals.

It was realized that most TMS functions would keep control of the computer or at least keep control of the TMS partition for the whole time that they were executing; thus most TMS modules needed only to be serially reuseable. For those situations where conflict would exist between two or more user programs attempting to use the same serially reuseable resource a simple queuing scheme employing a single chain of TMS "Function Blocks" was devised. (See page 8 for a discussion of Function Blocks [FB's].)

1.2 Basic Structure of the System

To the application programmer, the Terminal Monitor System (TMS) is 1) a collection of tables or blocks linked to each other and containing all information about the system; and 2) a set of processing routines which operate on these tables and programmer-supplied parameters to perform the necessary system services. Another part of the system is a top-level supervisor, the phantom job, which handles certain basic console operations. The application programmer's only interface with the phantom job is the fact that the phantom job calls the application program as a subroutine of itself.

There is a monitor routine for each of the major functions performed by TMS, plus some routines not directly accessed by the application program. Interface between the program and the monitor is accomplished by the use of special TMS macro instructions.

The basic block upon which the rest of TMS depends is the Communication Region (CR). To the application programmer, the prime use of this block is as a vector of entry point addresses to the monitor routines. There is only one CR for the entire system.

The basic block for each terminal is the Function Block (FB). This block contains such things as: an auxiliary save area for the application program associated with the terminal; the user identification code for the user logged in at that terminal; the application program name; the terminal number; several bytes of flags denoting terminal type and terminal and program status; pointers

to other control blocks in TMS; terminal I/O information and work areas; and pointers to chains of the main storage blocks and OS Data Control Blocks associated with that particular terminal.

For each terminal attached to the computer there exists a buffer for terminal input/output. For terminals that share the same communications link, these buffers are chained together and shared between those terminals. Thus it is not safe to assume that a particular I/O buffer address will always be associated with the application program during the period that it is in operation.

1.3 Services Provided

The Terminal Monitor System provides several services to facilitate the writing of application programs that will operate in a multi-programmed environment interacting with their users via remote terminals connected to the computer by communications lines and employing, if necessary, direct access data sets. These services generally take the form of one or more TMS macro instructions, which set up parameters and then transfer control to a suitable portion of the TMS monitor. The necessary tables to control the system and each application program running under it are also provided. Finally TMS provides an elementary top-level terminal control language to enable the housekeeping functions of user "log in" and specified application program loading to be performed.

There are several operations that each application program must perform in interfacing with the TMS such as obtaining new save areas, linking save areas, establishing base registers, and maintaining, if desired, pointers to various system tables. The macros provided for this purpose are TMSSAVE and TMSRETN.

Central to the operation of a multi-programming system is the ability for an individual application program to indicate that it no longer requires control of the computer until some asynchronous operation is completed, such as input/output. In TMS this function as well as the function of synchronizing program operation with these asynchronous operations is performed by the TMSWAIT macro instruction.

In a dynamic multi-programmed environment such as TMS, careful management of the main storage of the computer is necessary. In order to free system resources as soon as possible after it is determined that the application program no longer needs them (especially in the case where the user has lost control of his application program) it is also necessary to keep careful record of what areas of main storage are allocated to which application program. Both of these functions are embodied in the TMSGETM and TMSFREEM macro instructions which are used for obtaining and releasing main storage, respectively.

The Terminal Monitor System provides access to bodies of external data by any of the three basic access methods for data provided under OS; Sequential, Direct, and Indexed. In addition, certain functions of the Queued Indexed Sequential Access Method are simulated by TMS via the TMSSETL, TMSGET, and TMSESETL macro instructions.

It is desirable both to simplify the manner of obtaining access to data sets and to avoid the kind of errors that may result in the entire monitor run being abnormally terminated. In addition, the tables associated with data sets are often one of the leading obstacles to writing re-entrant programs. The TMSOPEN macro instruction combines the function of both the OS DCB and OPEN macro instructions. This macro instruction actually generates a DCB in specially obtained main storage and returns the address of the opened DCB to the user program. The corresponding macro TMSCLOSE is responsible for closing the data sets and freeing those areas of core obtained for the DCB and associated buffers.

2. SYSTEM CONVENTIONS

2.1 Use of Save Areas and Initial Base Registers

In the use of save areas and initial base registers the Terminal Monitor System conforms almost exactly to the standards of OS/360. Upon entry to the application program a save area, which is to become the top of a save area chain, is pointed to by register 13 (RS). Subsequent save areas must be linked to this save area using OS conventions and register 13 must point to a valid save area at all times. The only variance from standard OS practice is that the first word of each save area contains the address of the associated FB. As each save area is obtained, its first word must therefore receive the contents of the first word of the preceding save area. Code for maintaining this convention and the linkage between save areas is generated as part of the expansions of the TMSSAVE and TMSRETN macro instructions. These macro instructions also contain provisions for either providing an in-line save area, obtaining additional core storage for a save area, or suppressing the generation of any save area at all.

As in all OS programs, register 15 (RC) contains the entry point address when control is passed to the user program. The TMSSAVE macro instruction generates code to move this address into a permanent base register of the user's choosing. The default base register is register 12 (RB).

When it is necessary to make maximum usage of every register available, the application programmer may wish to use register 13 as a pointer to a general work area, of which the first 72 bytes are reserved for the save area. In this case he may specify SA=REMOTE and employ the SAINCR operand to specify the number of additional bytes he wishes obtained. If this increment plus the 72-byte save area is greater than 4095 bytes, a level 4 warning message will be issued, but the necessary code will be generated. In this case, it is the application programmer's responsibility to provide additional base registers as needed.

2.2 Use of FB and CR

Upon entry to an application program from the top level monitor, a pointer to the program's Function Block (FB) is provided in register 11 (R9) and a pointer to the general Communication Region (CR) is provided in register 10 (R8). Certain TMS macro instructions generate code to alter byte settings in the FB; the FB may also be used to locate the CR if a pointer to the CR is not being maintained by the user program. The CR is principally used by application programs as a vector of addresses of monitor routines.

With respect to the FB, the application programmer has the option of stating whether the program will or will not maintain register 11 as the FB pointer. The application programmer may encode RFB=NONE in the TMSSAVE macro instruction to indicate that register 11 is not guaranteed to contain a pointer to the FB at all times. If this is not coded, register 11 must point to the FB whenever any code generated by a TMS macro instruction is being executed. With respect to the CR pointer, the application programmer may specify any register (including register 10) to be the CR pointer by using the RCR operand of the TMSSAVE macro instruction. If this operand is omitted, it is assumed that no CR pointer will be maintained by the program. If the application programmer does specify a register as a CR pointer, he must insure that it contains the proper address whenever code generated by a TMS macro instruction is executed.

A set of standard register equates and symbolic definitions for the FB and CR are maintained on ILR.MACLIB. These may be obtained by the COPY REGS, COPY FB, and COPY CR statements, respectively. The register equate, FB, and CR definitions must be provided for every program that employs any TMS macro instruction.

2.3 Obtaining and Releasing Main Storage

In order to prevent system crashes due to running out of main storage and to properly clean up after an application program failure, TMS must maintain control over all main storage allocations within its partition. The macro instructions used to obtain and release main storage are almost identical to the R-type GETMAIN and FREEMAIN macro instructions in OS.

To obtain core storage, the TMSGETM macro instruction is used. Its only parameter is either a symbolic expression representing the number of bytes to be obtained or the designator, in parentheses, of a register which contains the count of bytes desired. The symbolic expression form of operand may be used only to request 4095 bytes or less. Upon return from TMS, register 1 (RP1) points to the address of the storage obtained.

The macro instruction used to release main storage, TMSFREEM, differs from FREEMAIN, its OS counterpart, in that it requires only one operand; the address of the area of main storage to be released. This parameter is supplied as either the symbolic address of a full word of storage containing the address of the area to be released, or the designator, in parentheses, of a register containing the address of the area to be released. The length of the area to be released is obtained by TMS from a link element that it maintains. This approach results in the restriction that any area of main storage obtained by TMSGETM macro instruction must be released by a corresponding TMSFREEM macro instruction; i.e., no area of main storage obtained as a single unit may be released in segments.

2.4 Opening and Closing Data Sets

The operations of defining, opening, and closing data sets in TMS are considerably different than the corresponding operations in OS. TMS takes advantage of the fact that all data sets are already defined and allocated on direct access storage devices with complete information in their data set control blocks (DSCB's). Thus TMS has no need for an analogue for the OS DCB macro instruction. Instead the few parameters needed to complete the definition of a data set are supplied as additional operands in the TMSOPEN macro instruction.

For direct access (DSORG=DA) data sets the following options are supported: READ, WRITE, CHECK, searching by block identification, and searching by key. For indexed sequential data sets (DSORG=IS) the following options are supported: READ, WRITE, UPDATE, and CHECK. The GET, SETL, and ESETL macros are simulated by corresponding TMS macro instructions. TMS does not presently support general access methods for update so there exist no analogues for PUT or PUTX. For partitioned data sets (DSORG=PO) the following options are supported: READ and WRITE (NOTE and POINT are implied). For sequential data sets (DSORG=PS) the following options are supported: READ, WRITE, CNTRL, and POINT.

A data set in TMS is both defined and opened by use of the TMSOPEN macro instruction. Several of the operands of this macro instruction have the same purpose and meaning as their counterparts in the OS DCB macro instruction. These include DSORG, MACRF, EODAD, OPTCD, SYNAD, EOEA, PCIA, SIOA, CENDA, and XENDA. Of the foregoing only the DSORG and MACRF operands are required. Since TMS speaks strictly in terms of data sets, the operand DSNAME is used in TMSOPEN instead of the operand DDNAME that is used in the OS DCB macro instruction. This operand, which is required, specifies the major portion of the data set name. Its parameter consists of from 1 to 8 EBCDIC characters with no embedded blanks or periods. In searching for the data set this name will be appended to the qualifier ILR., and may be further qualified by user or terminal-dependent information. This additional qualification is controlled by the QUALIFY keyword operand; some form of additional qualification must be supplied for any output data set. Specifying QUALIFY=BYNAME causes a period followed by the user's identification code to be appended to the data set name. This should be used for data sets which must be keyed through an individual user (such as DISCUS restart files). By specifying QUALIFY=BYFBNO the problem programmer causes the characters ".FB" to be appended to the data set name followed by the two digit terminal number (which is unique for every terminal in the system). This form of qualification is most useful for data sets which are not to be associated with a particular user but which must be guaranteed to be unique within the system at any given moment (such as off-line print files).

In addition to obtaining storage for, and creating, the user's DCB, the system will proceed to obtain storage for 1 buffer and insert the proper pointers into the DCBBUFCB address pointer. DCBBUFCB will point to a standard OS 8-byte buffer control block followed immediately by one buffer of the proper length as indicated in the BLKSIZE information stored in the DSCB. The acquisition of this buffer by TMS may be suppressed by encoding the operand BUFFERS=NO, in which case the program will be responsible for supplying its own buffer area.

Certain information usually supplied as OPEN macro instruction parameters in OS is supplied in TMS by use of the FOR keyword operand. This operand indicates for what type of processing the data set is being opened. The permissable forms of this parameter (not all of which apply to every data set organization) are INPUT, INOUT, OUTIN, OUTPUT, and UPDAT. Each of these parameters has the same result as specifying the corresponding parameter in an OS OPEN macro instruction.

The problem programmer has some measure of control over the processing of any errors that occur during the data set location and opening process. The default option is to go to TMSPURGE to terminate the entire program. By specifying the RETURN=YES operand, the user programmer may receive control back from the system with an appropriate completion code in register 15 (RC).

For all forms of data set organization, the EXCP access method may be specified with or without appendages.

2.5 Terminal I/O

Communications between an application program and a remote terminal are handled by the macro TMSCSIO. This macro supports several different methods of specifying messages for output, the basic read and write operations to the terminal, a set of device-dependent operations (such as screen erase or typewriter carriage return) and the choice of an immediate or deferred wait.

2.5.1 Write Operations

The message to be written is specified as the first positional operand in one of three ways: 1) the message text itself may be provided, enclosed in apostrophes; 2) a symbolic address of the message may be provided; or 3) the designator (enclosed in parentheses) of a register which contains the address of a message may be provided. The fact that this is a write operation is specified by encoding WRITE as the first subparameter of the OP parameter. The second and succeeding subparameters indicate additional device dependent operations to be performed with the write. For cathode ray tube (CRT) devices these include EBW for erase screen before write, and/or NL for the new line before write. For mechanical typewriter-type terminals RBW for carriage return before write

and/or RAW for carriage return after write may be specified. Device-dependent operations for both CRT's and typewriter-type terminals may be specified in the same write operation; the system will determine which type of terminal is being serviced and ignore any operation specified for the other type of terminal. This feature may be used to obtain a measure of device independence in problem programs.

Messages to be written may appear in one of two formats. Format 1 consists of a half-word count of the number of characters in text followed immediately by the text itself. Format 2 consists of the text alone; a separate character count is provided elsewhere. If the message is a Format 2 message, its length must be provided in the LENGTH operand. The LENGTH parameter may be either a symbolic expression whose value is the message length (1,021 bytes or less) or a register designator (in parentheses) of a register which contains the length count. Absence of any LENGTH operand indicates a Format 1 message.

2.5.2 Read Operations

Read operations are specified by coding READ as the first positional subparameter of the OP parameter. Upon conclusion of the input-output operation, register 1 (RP1) points to the first character of the text read. Register 0 (RP0) contains a count of the number of characters of incoming text. This count includes the end of transmission (EOT) character which appears as the last non-blank character in the buffer.

2.5.3 Additional Features

The problem programmer may overlap console I/O with other operations if he desires. Use of the WAIT=DEFER operand allows processing to continue simultaneously with console input-output. A separate TMSWAIT macro instruction must be coded by the application programmer to wait for the completion of console I/O. Coding WAIT=IMMED causes a call to TMSWAIT to be generated immediately following the call to the console I/O routine: this is also the default option. If it is desired that return be made to a point other than the instruction immediately following the TMSCSIO macro instruction, the programmer may provide a symbolic address by using the RET keyword operand. Transfer is then made to this address upon return from TMS.

2.6 Non-Terminal I/O

Input/output to user's data sets is accomplished by using the standard OS macro instructions with the restriction that only those macro instructions and/or operands that are supported by the TMS system may be employed. (For a discussion of which access methods are supported, see "Opening and closing data set" above.) The only exceptions to this rule are the following: (1) TMSWAIT must

be used in place of any WAIT or WAITR macro instruction; (2) if an OS CHECK macro instruction is employed the macro instruction TMSWAIT must immediately precede it, specifying the same DECB.

2.7 Scheduling and Event Synchronization

The release of the partition so that other TMS users may use the CPU and the synchronization of program processing with I/O events is accomplished in a way completely analogous to that of OS. In order that TMS may maintain complete control of the situation and make maximum utilization of the available resources, a special system macro called TMSWAIT is employed. The ECB keyword operand provides the symbolic address of the event control block on which the application program is to wait. The default specification is FBECB, which is used for waiting on terminal I/O operations. An optional keyword operand RET may be used to provide the symbolic address of the next instruction to be executed following return from the wait. Absence of this operand indicates that the next instruction to be executed is that immediately following the TMSWAIT macro instruction.

Certain forms of terminal I/O require processing following the completion of the wait for I/O to complete. Thus, another keyword operand, OP, is used to indicate what form of console I/O we are waiting on. The permissible parameters for this operand are READ, WRITE, CLEAR, and REWRITE. Depending upon TMS requirements, additional code may be generated following the branch to the wait routine.

3. SPECIFICATIONS FOR CODING TMS MACROS

3.1 TMSCLOSE -- Close a Data Set

The TMSCLOSE macro instruction causes the specified data set to be closed and the main storage for the DCB and buffers (if supplied by TMSOPEN) to be released.

[symbol]	TMSCLOSE	DCB = (data control block register) / data control block address

Data control block register

is the symbolic or literal definition of a register which contains the address of the data control block to be closed. If (1) or (RP1) is designated, register 1 must contain the DCB address prior to invoking the macro instruction.

Data control block address

is the symbolic address of a full-word which contains the address of the data control block to be closed.

3.2 TMSCSIO -- Terminal Input/Output

TMSCSIO macro instruction causes an input or output operation to be initiated at the terminal associated with the function block pointed to by register 11 (RFB). Depending upon the parameters of this macro instruction, a separate TMSWAIT macro instruction may be necessary to test for completion of the operation.

[symbol]	TMSCSIO	[('message' / (message register) / message address)], OP = (READ / WRITE / CLEAR / REWRITE [,option],...) [,LENGTH= (length register / length expression)] [,WAIT= (IMMED / DEFER)] [,RET=return address]

Message

is the message to be transmitted by a WRITE operation. The LENGTH parameter must not be coded.

Message register

is the symbolic or literal definition of a register which contains the address of a Format 1 or Format 2 message which is to be transmitted by a WRITE operation. If (1) is designated, register 1 must contain the address of the message. (See section 2.5 for description message formats.)

Message address

is the symbolic address of a Format 1 or Format 2 message which is to be transmitted by a WRITE instruction.

OP = READ - read from the terminal
WRITE - write to the terminal
CLEAR - erase screen (CRT's only)
REWRITE - no meaning at present (for planning purposes only).

Option

is one of the following:

EBW - Erase the contents of the screen before writing (CRT's).
RAW - Return the carriage after transmitting the message (Typewriters).
RBW - Return the carriage before transmitting the message (Typewriters).
NL - Start the message on a new line (CRT's).

Length register

is the symbolic or literal designation of a register which contains the length of a Format 2 message. If (0) is designated, register 0 must contain the length of the message.

Length expression

is any expression suitable for use in an LA instruction which represents the length of a Format 2 message.

WAIT=IMMED

causes generation of a call to the wait routine as part of the macro expansion.

WAIT=DEFER

causes no generation of a call to the wait routine. A separate TMSWAIT macro instruction is necessary.

Return address

is the symbolic address to which control is to be returned after execution of the macro instruction. If omitted, control passes to the next instruction in sequence.

3.3 TMSDLETE -- Delete a load module

The TMSDLETE macro instruction causes the responsibility count for the load module specified to be decreased by one, and when the responsibility count reaches zero the core occupied by the module is released and the request block dropped from the load list. The module must have been loaded via the TMSLOAD macro instruction.

The corresponding user load list is updated, and the area occupied by the load list element for this module is freed.

If the request specifies a name that is not on the user load list, the requesting program is purged.

The TMSDLETE macro instruction is coded as follows:

[symbol]	TMSDLETE	{ EPNAME = entry point name EPLOC = address }

where:

'entry point name'

is the legal entry point name as specified in the TMSLOAD macro instruction.

'address'

is the symbolic address (or register containing this address) of an entry point name.

3.4 TMSESETL -- End Sequential Retrieval (QISAM input only)

The TMSESETL macro instruction ends the sequential retrieval of data from an indexed sequential data set, and causes the buffers associated with the specified data control block to be released. A TMSESETL macro instruction, or an end of data set indication must separate TMSESETL macro instructions issued for the same data control block.

The TMSESETL macro instruction is written as follows:

[symbol]	TMSESETL	dcb address

dcb address

is the address (or register specification containing this address) of the data control block for the indexed sequential data set being processed.

3.5 TMSFREEM -- Release Main Storage

The TMSFREEM macro instruction causes a block of main storage obtained by TMSGETM to be released.

[symbol]	TMSFREEM	A = (storage address register).

Storage address register

is the symbolic or literal definition of a register containing the address of the block of main storage to be released.

3.6 TMSGET -- Obtain Next Logical Record (QISAM input)

The TMSGET macro instruction causes the monitor system to retrieve the next record and to return the main storage address of the record in register 1. Control is not returned to the problem program until the operation is complete.

The TMSGET macro instruction is written as follows:

[symbol]	TMSGET	{ dcb address (address register) }

dcb address

is the address of the data control block for the data set being retrieved.

(address register)

is the specification in parentheses of a register containing the data control block address.

NOTE:

When control is returned to the problem program, register 0 should be tested for a completion code as follows:

If Reg 0 is:	Reg 1 points to:
F'0'	Logical Record
F'4'	message: "Record with specified key does not exist'
F'8'	message: ' I/O errors"

NOTE:

For QISAM under TMS, the TMSOPEN macro instruction must specify BUFFERS=NO.

3.7 TMSGETM -- Obtain Main Storage

The TMSGETM macro instruction causes a variable-length block of main storage to be allocated to this terminal. The address of this block is returned in register 1. The programmer may specify that control passes back to the calling program in the event of error.

[symbol]	TMSGETM	LV = {(length value register) / length value expression}, [RETURN={YES / NO}]

Length value register

is the symbolic or literal definition of a register which contains the size in bytes of the desired block of main storage.

Length value expression

is an expression whose value represents the size in bytes of the desired block of storage. When specified in this manner, the size must not be greater than 4095 bytes.

RETURN = NO

indicates that control passes to the PURGE routine of the monitor if insufficient main storage is available.

RETURN = YES

indicates that control returns to the calling program under all circumstances. The return code in register 15 is as follows:

Return Code	Meaning
0	The main storage requested was allocated.
4	No main storage was allocated.

3.8 TMSHDCPY -- Provide Hard Copy of CRT Output

The TMSHDCPY macro instruction provides the facility to obtain a permanent copy of console I/O operations via the use of the 1403 line printer.

The output to the printer recognizes all carriage control characters inherent in the data and as many of the format characters as feasible.

The TMSHDCPY macro instruction is coded as follows:

[symbol]	TMSHDCPY	MES = address, LENGTH = {(register) / address}

where:

MES = address

'address' is the symbolic specification of the address (or register containing this address) of the start of the text to be output. If this operand is omitted, the text on the entire screen will be read and copied.

LENGTH = address

'address' is the symbolic or absolute specification of the length of the text to be output. '(address register)' is the specification, in parentheses, of a register containing this length. If MES is not coded, this operand need not be.

The TMSHDCPY routines will return a return code in register 15 as follows:

0 - processing completed normally

08 - processing partially completed, not enough core

12 - no processing done, not enough core

3.9 TMSLOAD -- Bring a Load Module into Main Storage

The TMSLOAD macro instruction causes the monitor system to bring the load module containing the specified entry point into main storage if a usable copy is not available. The responsibility count for the load module is increased by one. Control is <u>not</u> passed to the load module; instead, the main storage address of the designated entry point is returned in register 0. The load module remains in main storage until the responsibility count is reduced to zero through the use of the TMSDLETE macro instruction.

The TMSLOAD macro instruction is written as follows:

[symbol]	TMSLOAD	{EPNAME = symbol EPLOC = address of name}

EPNAME =

is the entry point name in the load module to be brought into main storage.

EPLOC =

is the main storage address of the entry point name described above. The name will be padded with blanks to eight bytes if necessary.

When control is returned from the TMSLOAD macro instruction, register 15 contains a return code as follows:

0 - successful load

4 - not enough core to load program

8 - module missing from library

12 - module non-reentrant and already loaded

16 - I/O error reading module directory

3.10 TMSLOG -- Create an Entry in the System Log

The TMSLOG macro instruction will create an entry in the system log for the requesting program. It is expected that the program will want to create information of its own within this entry; therefore, provision is made for up to 2024 bytes of user information to be written.

Each entry into the log is prefaced by a 16 byte leader as follows:

Bytes:	2	2	4	8
Field:	LEN	FBNO.	FBNAME	FBPNAME

where:

LEN is the total length of the entry including header

FBNO. is the number of the terminal which had control when the request was issued

FBNAME is the name of the user logged in on that terminal

FBPNAME is the name of the program that was requested by the terminal user (not necessarily the name of the program issuing the macro)

Records are written RECFM=V and the BDW and RDW created by the system.

The TMSLOG macro instruction is coded as follows:

[symbol]	TMSLOG	{'message' / message address}, LEN = number

where:

'message'

is a literal string of the actual entry to be created

message address

is the address (or the specification in parentheses of a register containing the address) of the log entry to be created.

number

is the absolute or symbolic designation of the length (or the specification in parentheses of a register containing this length) of the entry to be created. Do not include length of leader. If this operand is omitted, or set to zero, only the 16 byte leader is written.

Only the combinations listed above are valid for the TMSLOG macro instruction.

Upon return, register 15 (RC) contains a return code as follows:

0 - normal completion.

08 - DCB has not been opened (report to TMS system consultant).

12 - length supplied by processing program was invalid.

16 - not enough core was available for creation of the entry.

3.11 TMSOPEN -- Generate Data Control Block and Open a Data Set

The TMSOPEN macro instruction causes the system to obtain core for and generate a data control block for the data set named. Unless suppressed, one buffer is also provided preceded by a standard buffer control block pointed to be DCBBUFCB. The program may specify that it is to receive control with a completion code in register 15 (RC) in the event of an error. Upon normal completion, the

newly-generated DCB is pointed to by register 1 (RP1). If BUFFERS=YES was specified, register 0 (RP0) points to the buffer provided.

[symbol]	TMSOPEN	DSNAME = data set name, DSORG = data set organization code, MACRF = (macro reference code [,...]) [,OPTCD = (option code)] [,SYNAD = synchronous error address] [,EODAD = end-of-data address] [,FOR = {INPUT / OUTPUT / INOUT / OUTIN / UPDAT}] [,BUFFERS = {YES / NO}] [,QUALIFY = {BYNAME / BYFBNO}] [,RETURN = {YES / NO}]

Data set name

is a one- to eight- character name that will become part of the data set name used to search the system catalog and disk volume tables of contents for a corresponding pre-allocated data set. The basic set name generated will be of the form:

ILR. nnnnnnnn

where nnnnnnnn represents the name supplied as this parameter.

Data set organization code

is a two-or-three-character code representing the organization of the data set. The permissable codes in TMS are:

DA,DAU - Direct access

IS - Indexed sequential

PO,POU - Partitioned

PS,PSU - Sequential

Macro reference code

is one of the combinations below, depending on data set organization and access method:

BSAM

```
    C
(R  P  )
    C
(W  P  )
(R [C],W[C])
(R [P],W[P])
```

Character	Definition
C	CNTRL
P	POINT (implies NOTE)
R	READ
W	WRITE

BDAM

```
    K
(R  I   [C])
    KI
    K
(W  I   [C])
    KI
                  K
    K             I
(R  I   [C]  ,W       [C])
    KI
                  KI
```

Character	Definition
C	CHECK (absence denotes WAIT)
I	Search to be made by block identification
K	Search to be made by key
R	READ
W	WRITE

BISAM

```
(R  [C]  ) W
(W  [U]  [C])
(R  [U]  [C],W [U] [C])
```

Character	Definition
C	CHECK
R	READ (implies FREEDBUF)
U	Records are to be updated. If U is coded with R, U must be coded with W.

QISAM

```
{(GL[,S[k]])}
```

Character	Definition
GL	GET (Address of buffer to be provided by the control program
K	Search to be made by record or generic key
S	SETL
(NOTE:	For QISAM, TMSOPEN must specify BUFFERS = NO)

[ ] indicates optional character; select one from vertical stack within { }; select one or none from vertical stack within [].

BPAM

$\left\{\begin{matrix}(R)\\(W)\\(R,W)\end{matrix}\right\}$

Character	Definition
R	READ (implies NOTE and POINT)
W	WRITE (implies NOTE)

Option code

specifies optional services to be performed. These are a subset of the services available under OS. The permissable combinations in TMS are:

BDAM

[W] [E] [F] $\left[\begin{matrix}R\\A\end{matrix}\right]$

Character	Definition
A	Specifies that actual device addresses are to be presented ("block address" operand) in READ and WRITE macro instructions.
E	Requests an extended search (more than one track) for block or available space. Ignored if A is also coded. Refer to IBM manual "Supervisor and D.M. Macro Inst." for the discussion of the LIMCT operand for a description of extended search.
F	Specifies that when feedback is requested in a READ or WRITE macro instruction, the device address returned is to be of the form presented to the control program. If F is omitted, feedback is in the form of the actual device address of the block.
R	Specifies that relative block addresses are to be presented ("block address" operand) in READ and WRITE macro instructions.
W	Requests a validity check for write operations. If the device is a 2321 data cell, validity checking is always performed, whether requested or not.

[ ] indicates optional character; select one or more from vertical stack within [ ].

If neither R nor A is coded, relative track addresses are assumed.

BPAM	Character	Definition
W	W	Requests a validity check for write operations. If the device is a 2321 data cell, validity checking is always performed, whether requested or not.

BSAM	Character	Definition
U W WC	U	Printer with Universal Character Set feature only -- unblocks data checks and allows analysis by an appropriate error analysis (SYNAD) routine. If U is omitted, data checks are blocked (not recognized as errors).
	W	Direct access device only -- requests a validity check for write operations. If the device is a 2321 data cell, validity checking is always performed, whether requested or not.

Synchronous error address

has the same function as in the OS DCB.

End-of-data address

has the same function as in the OS DCB.

FOR = INPUT

indicates an input data set.

FOR = OUTPUT

indicates an output data set.

FOR = INOUT

indicates an input data/set initially and, without reopening, an output data set.

FOR = OUTIN

indicates an output data set initially and, without reopening, an input data set.

FOR = UPDAT

indicates a data set to be updated in place.

BUFFER = YES

indicates that TMS is to provide a single buffer of the necessary length.

BUFFERS = NO

indicates that the application program will provide its own buffers.

QUALIFY = BYNAME

indicates that the final data set name generated will be:

ILR.nnnnnnnn.xxxx

where nnnnnnnn is the name specified by the DSNAME operand and xxxx is the user identification code for the current user of the program.

QUALIFY = BYFBNO

indicates that the final data set name generated will be:

ILR.nnnnnnnn.FBxx

where nnnnnnnn is the name specified by the DSNAME operand and xx is the terminal number for the current user of the program.

RETURN = YES

indicates that in case of an error during TMSOPEN processing, control is to be returned to the calling program. The contents of register 15 (RC) indicate results of the OPEN as follows:

Return code	Meaning
0	Successfully located and opened.
4	Unable to locate the data set in the system catalog or in the volume table of contents of all DASD's on which it resides.
8	Insufficient core remains to complete open processing.
12	Disastrous error during open processing.

RETURN = NO

indicates that in case of an error, control is to be returned to the monitor.

3.12 TMSPRINT -- Put a line to line printer

The TMSPRINT macro instruction causes a line to be printed on the off-line printer. All output using this macro instruction is clearly identified by a header preceding it.

[symbol]	TMSPRINT	{ area address / (address register) }

where:

area address

is the address of a 133 byte line to be printed, whose first character is the ASA printer control character. (see Appendix 3);

(address register)

is the specification of a register which contains the address of the line to be printed.

3.13 TMSRETN -- Return to calling program

The TMSRETN macro instruction causes the restoring of the registers which were saved by TMSSAVE when the program was entered. Control is then returned to the calling program, (i.e., the monitor). If TMSSAVE obtained main storage for a new save area, then TMSRETN will release this storage.

[symbol]	TMSRETN

3.14 TMSSAVE -- Entry from calling program

The TMSSAVE macro instruction causes the establishment of a control section with the symbol in the name field being used as the control section name. It generates code to save all registers in a standard save area, establish a new save area if desired, and establish a base register. If a register is designated as a communication region pointer, code is generated to load that register. If a remote new save area is called for, code is generated to obtain core storage for that save area; otherwise, the new save area is generated in-line.

[symbol]	TMSSAVE	[RBASE= base register] [,RFB=NONE] [,RCR=CR pointer register] [,RWORK= work register] [,SA={LOCAL / NONE / REMOTE}] [,SAINCR= length increment]

Symbol

is the name of the control section and entry point of the user program.

Base register

is the symbolic or literal designation of a register to be used as the first base register. If omitted, register 12 (RB) is assumed.

RFB = NONE

indicates that the program will not maintain register 11 (R9) as the pointer to the associated FB.

CR Pointer register

is the symbolic or literal designation of a register that will contain the address of the communication region whenever a TMS macro instruction is invoked. If omitted, no such register is established and all TMS macro instructions generate slightly slower code. Upon entry to the user program, register 10 (R8) contains the communication region pointer.

Work register

is the symbolic or literal designation of a register that will be used as a work register for establishing save area linkage. If omitted, register 2 (R0) will be used.

SA=LOCAL

indicates that the new save area is to be generated in-line in the expansion of TMSSAVE. This makes the user program non-reentrant.

SA=NONE

indicates no save area provided. Programmer must provide his own.

3.15 TMSSETL -- Set Lower Limit of Sequential Retrieval (QISAM input only)

The TMSSETL macro instruction causes the monitor system to start processing the input request at the specified record. Sequential retrieval of records using the TMSGET macro instruction continues from that point until the end of the data set is encountered or a TMSCLOSE or TMSESETL macro instruction is issued. A TMSESETL macro instruction must be issued between TMSSETL instructions that specify the same data set.

The TMSSETL macro instruction can specify that retrieval is to start at the beginning of the data set, at a specific record, or at the first record of a specific class of records.

The TMSSETL macro instruction is written as follows:

[symbol]	TMSSETL	{dcb address / (address register)}	{B / K, key address / KC, key address, length}

dcb address

is the address of the data control block for the data set being retrieved

(address register)

is the specification in parentheses of a register containing the data control block address

B specifies to start at the beginning of the data set

K specifies starting at the record with the specified key

KC retrieve the first record of a specified key class

key address

is the symbolic name of a main storage location (or a register containing this address) of the key of the record wished. In the case of type KC, this is the address of the partial key specifying the key class

length

is the length of the partial key given for the key class. This may be an absolute decimal number, or the specification of a register containing the value right adjusted in binary.

3.16 TMSWAIT -- Wait for an event

The TMSWAIT macro instruction causes the user program to be dismissed until completion of the event associated with the designated event control block. Only one event control can be waited on at a time by any user program.

[symbol]	TMSWAIT	[ECB = event control block] [,OP = operation code] [,RET = return address]

Event control block

is the symbolic address of an event control block which conforms to all the rules of a standard OS event control block. If omitted, FBECB (the terminal input/output event control block) is assumed.

Operation code

is one of the following:

READ - Upon completion of the wait, the terminal text length (including EOT character) is in register 0 (RP0) and the text address is in register 1 (RP1).

WRITE - No effect.

CLEAR - No effect.

REWRITE - No effect.

Return address

is the symbolic address to which control is to be returned after execution of the macro instruction. If omitted, control passes to the next instruction in sequence.

4. THE TOP-LEVEL CONTROL LANGUAGE

In order that TMS may deal with a great many users and offer each user the choice of which of many user programs he wishes to operate under, a minimum form of executive program is necessary. This program exercises complete control over the operation of the system and interfaces with the terminal user via the top-level control language. Among the services provided by this top level supervisor are: the identification of a terminal user via the process of "logging in"; the acceptance of a user program name, and the loading of the associated user program; the notification of the user when certain errors have occurred; and the ability to logically disconnect the individual terminal from the computer when necessary.

4.1 Logging In

When TMS first begins operation the following message is directed to each terminal:

TMS100I TMS IN OPERATION

followed immediately by the message:

TMS101A WAITING FOR LOGIN

This latter message indicates that the supervisor is waiting for the user to identify himself at the terminal by typing in a user identification code of up to four characters (this code will have been assigned to each user by laboratory supervisory personnel). The proper response to any request for input from the top level supervisor varies with the various types of terminals. For the Sanders Associate CRT displays the following sequence must be followed: push the CLEAR button followed by the FORMAT TYPE button then type in the desired response (in this case the user login code) and depress the SEND BLOCK button. There are several responses to an attempt to login. The message:

TMS102I NOT ACCEPTED

indicates that the user identification code provided is not accepted as valid. The message:

TMS103I NOT ACCEPTED, NAME ALREADY IN USE

indicates that while the user identification code supplied is valid, a user at another terminal is already "logged in" under this code. After either of the above two messages the message:

TMS101A WAITING FOR LOGIN

is reissued inviting another attempt to "log in". If the user

identification code has been accepted as valid the message:

TMS102I NAME LOGGED IN

is returned where name is the user identification code as recognized.

4.2 Specifying Programs

Immediately after the message accepting the login the message:

TMS104A SPECIFY PROGRAM

will appear. This indicates that the user is to respond with the name of the user program that he wishes to have loaded and assigned to his terminal. The names and purposes of the individual user programs are specified in a separate publication; each name will be at least 1 but no more than 8 characters in length. There are several messages that may indicate difficulty in loading the requested program. The message:

TMS107I PROGRAM NOT FOUND

indicates that the name supplied is not the name of a program currently in the TMS library. The message:

TMS108I PROGRAM NON-REENTRANT AND ALREADY IN USE. WAIT OR TRY ANOTHER

has a more complex meaning. This message indicates that the program does exist but that it may only be used by one terminal at a time and is already being used at another terminal. The user should wait until the other user has exited the program and repeat his request for the program or he should request another program that he can use in the interim. The final error message that may appear is:

TMS109I NOT ENOUGH CORE TO LOAD PROGRAM

This indicates that insufficient main storage remains in the computer to load the executable code of the program requested. Each of the above three messages is immediately followed by the message:

TMS104A SPECIFY PROGRAM

which invites the user to try again. In the event that the program is successfully loaded the next message will be generated by the user program itself. To understand the purpose of such messages users should consult the documentation of the individual user programs. Depending upon the individual user program and assuming that no errors have occurred in the operation of the user program in the interim the user will specify in some manner that

he wishes to exit from that user program and return to the top level supervisor. If this is done properly the message:

TMS106I NORMAL EXIT FROM USER PROGRAM

will appear followed by a report of the message:

TMS104A SPECIFY PROGRAM

this allows the user to specify a new program for execution or to leave the system.

4.3 Logging Out

The processing of indicating to TMS the user has finished at this particular time and wishes to leave the system is called "logging out". The user is able to log out at any time when the last message appearing at the terminal is:

TMS104A SPECIFY PROGRAM

Instead of specifying a program the user responds with the word LOGOUT. If this is correctly done the system will respond with the message:

TMS105I NAME LOGGED OUT

followed by the message:

TMS101A WAITING FOR LOGIN

which indicates that a new user may now sit down at the terminal and identify himself to the system.

4.4 Error Exits from User Programs

If TMS detects some form of error in the user program that threatens the integrity of the system it will force that user program to cease execution and return control to the supervisor. A series of messages will be returned to the terminal. The first one or two messages will generally be preceded by a TMS message identifier with a number in the range 150-199. These indicate the particular form of error encountered and are summarized in Appendix 2. These specific messages are then followed by the more general message:

TMS110I ABNORMAL RETURN FROM USER PROGRAM VIA PURGE ROUTINE

Receipt of this message verifies that an error has occurred and that the program has been terminated, all main storage obtained by the program has been released, and all data sets opened by the program have been closed. This message is followed by the

message:

TMS104A SPECIFY PROGRAM

which invites the user to restart the same program, start a new program, or "log out" from the system.

4.5 Special Operations

Under certain circumstances it becomes necessary to logically disconnect an individual terminal from the computer. This is best done only by persons with a thorough working knowledge of the system since at the current time this disconnection is irreversible. This operation may be performed only when the last message received from TMS is:

TMS101A WAITING FOR LOGIN

instead of responding with a user identification code the user responds with the word DISCONNECT. The terminal ceases operation at this point and no further message is received from the system. A confirmation is typed out at the computer; if there is any question about the success of the disconnect procedure the computer center operator may be contacted and requested to examine his console printout. When all terminals have been DISCONNECTed, the Terminal Monitor System will cancel itself.

APPENDICES

MACRO INSTRUCTION	OPERANDS			WRITTEN AS REGISTER				
		SYM	DEC DIG	(2-12)	(1)	(Ø)	RX TYPE	A-TYPE ADCON TYPE
TMSCLOSE	DCB =			X	X			
TMSCSIO	message	any message within apostrophes						
	OP =	refer to macro description						
	LENGTH =	X	X	X		X		
	WAIT =	refer to macro description						
	RET =	refer to macro description						
	MF =	refer to macro description						
TMSDLETE	EPNAME =						X	
	EPLOC =			X	X		X	
TMSESETL	dcb address			X	X		X	
TMSFREEM	A =	X		X	X			

MACRO INSTRUCTION	OPERANDS	WRITTEN AS						
		SYM	DEC DIG	REGISTER (2-12)	REGISTER (1)	REGISTER (Ø)	RX TYPE	A-TYPE ADCON TYPE
TMSGET	dcb address			X	X		X	
TMSGETM	LV =	X	X	X		X		
	RETURN =	refer to macro description						
TMSHDCPY	MES =			X	X		X	
	LENGTH =	X	X	X		X	X	
TMSLOAD	EPNAME =						X	
	EPLOC =			X	X		X	
TMSLOG	message	any message within apostrophes						
	message address			X	X		X	
	LEN =	X	X	X		X		
TMSOPEN		refer to macro description						
TMSPRINT	line address			X	X		X	

MACRO INSTRUCTION	OPERANDS	WRITTEN AS						
				REGISTER				
		SYM	DEC DIG	(2-12)	(1)	(ø)	RX TYPE	A-TYPE ADCON TYPE
TMSRETN		written as shown						
TMSSAVE	RBASE =	X		X				
	RCR =	X		X				
	RWORK =	X		X				
	SA =	refer to macro description						
	RFB =	refer to macro description						
	SAINCR =	X	X					
TMSSETL	dcb address			X	X		X	
	type	refer to macro description						
	key address			X		X	X	
	key length	X	X	X				
TMSWAIT	ECB =			X	X		X	
	OP =	refer to macro-instruction						
	RET =						X	

APPENDIX 2

List of System Messages

TMS 150 I - PROGRAM ENDED WITH STORAGE OR DATA SET STILL ATTACHED

System action: Close all DCB's left and free all storage.

Programmer response: Make sure that an exit from the program is not forced which leaves DCB's open or storage still attached.

TMS 151 I - INSUFFICIENT MAIN STORAGE LEFT TO SATISFY TMSGETM REQUEST

System action: User purged.

Programmer response: Wait until storage is freed by another program and try again, or reduce the amount of storage requested.

TMS 152 I - TMSFREEM REQUEST DOES NOT SPECIFY LEGITIMATE ADDRESS

System action: User purged.

Programmer response: Make sure the program has not incorrectly modified its storage pointers. If this happens consistently, the program is probably at fault; if erratically, report to system consultant.

TMS 153 I - ATTEMPT TO OPEN AN UNAVAILABLE/UNCATALOGUED DATA SET

The system has detected that the data set name specified in a TMSOPEN request is not catalogued or, if catalogued, is unavailable to the user because:

1. The data set is qualified;
2. The data set cannot be shared; or
3. The data set does not exist
4. An I/O error was discovered reading the catalogue or DSCB

System action: User purged.

Programmer Response: In case (1), catalogue the data set; in case (2) or (3), create or change the attributes of the data set. In case (4), report to the system consultant.

TMS 154 I - INSUFFICIENT MAIN STORAGE LEFT TO COMPLETE OPEN OF A DATA SET

System action: User purged

Programmer response: Wait until main storage is freed by another program and try again.

TMS 155 I - DISASTROUS ERROR IN TMS OPEN

The system has detected an unrecoverable error in the TMSOPEN processing.

System action: User purged.

Programmer response: Make sure that the parameters supplied in the TMSOPEN macro instruction are consistent with the attributes of the data set.

TMS 156 I - TMSCLOSE REQUEST DOES NOT SPECIFY LEGITIMATE ADDRESS

The system has detected that the address supplied in a TMSCLOSE request is not the address of a DCB which was obtained via a TMSOPEN macro instruction.

System action: User purged.

Programmer response: Make sure the program has not incorrectly modified the register or area containing the address of the data control block.

TMS 157 I - END OF DATA DETECTED WITH NO EODAD SPECIFIED

An end of data condition has been detected with no end of data address specified in the data control block.

System action: User purged.

Programmer response: Modify the TMSOPEN request to include the EODAD parameter.

TMS 158 I - SYNCHRONOUS ERROR DETECTED WITH NO SYNAD SPECIFIED

A synchronous error condition has been detected with no SYNAD exit specified in the data control block.

System action: User purged.

Programmer response: Modify the TMSOPEN request to include the SYNAD parameter.

TMS 159 I - INSUFFICIENT CORE LEFT FOR DEBUGGING

The user specified that he wanted to use the debugging facility, but not enough core was available for this facility to be used.

System action: User purged.

Programmer response: Wait until main storage is freed by another user and try again.

TMS 160 I - ERROR DETECTED IN TMS. USER PURGED WITH SNAP

The system has detected a program error within itself.

System action: User is purged with a snap dump.

Programmer response: none

TMS 161 I - ERROR DETECTED IN TMS. USER PURGED, SNAP UNSUCCESSFUL

The system has detected a program error in itself.

System action: User purged, snap dump was unsuccessful.

Programmer response: none.

TMS 162 I - ERROR DETECTED IN PROGRAM, USER PURGED

The system has detected a program error in the user program, but a debugging request was not specified.

System action: User purged.

Programmer response: modify program if error is known, or request debugging facility and try again.

TMS 200 I - ERROR OCCURRED AT RELATIVE LOCATION XXXXXXXX

This message occurs in conjunction with one of the above as information for the programmer only in deciding where in the program the error occurred.

APPENDIX 3

Carriage Control Characters

(1403 line printer)

b - Single space before printing

+ - No space before printing

0 - Double space before printing

- - Triple space before printing

1 - Eject to line 6 before printing

2 - Skip to next 1/2 page (line #9 or #36)

3 - Skip to next 1/3 page (line #8, #26, or #44)

4 - Skip to next 1/4 page (line #7, #21, #35, or #49)

6 - Skip to line 66 before concave paper fold

7 - Skip to line 66 before convex paper fold

8 - Skip to line 1

9 - Skip to line 61

All other characters are interpreted as blank.

PART II

A PROGRAM LOGIC MANUAL FOR THE TERMINAL MONITOR SYSTEM

1. DETAILED SYSTEM STRUCTURE

The structure of TMS closely parallels the structure of OS/360 and other complex operating systems implemented on 360-type machines. The system consists of a set of tables and a set of processing modules (or sub-programs). There are several types of tables, each containing information on the current state of the system itself, the application programs running under it, and the computer's resources. All tables can be located from other tables by a system of pointers. The processing modules have two principal tasks: to maintain and update the tables; and to request the IBM Operating System to perform certain tasks; (i.e., transmit a message to a particular terminal).

The remainder of Section 1 will discuss the tables peculiar to TMS and the interrelationships of these tables. Subsequent sections will introduce the processing modules and deal with their operation in detail.

1.1 Communication Region

The Communication Region (abbreviated CR) is, in essence, the key table of the system. As its name implies, it facilitates system communication by combining in one place pointers to virtually every other table and processing module in the system. The CR is important when a processing module must re-establish pointers to all systems tables, such as in processing I/O interrupts. The Communication Region may be located in one of two ways when no pointers exist: via the entry point TMSCRADR in the module TMSBEGIN; or by first locating a function block via word 0 of any save area and then using the back pointer to the CR found in the FB.

The CR contains several entry point addresses, primarily to those processing modules which may be accessed directly by application programs by use of TMS macros. These are TMSWAIT, TMSCSIO, TMSPURGE, TMSPLOAD, TMSGMFM, TMSSNAP, TMSOPEN, and TMSCLOSE. Several CR entries are used in wait list processing. These are a pointer to the wait list itself, a pointer to the current last entry, and a halfword containing a count of bytes in the basic wait list. A dummy Event Control Block is kept in the CR to indicate that a shared resource for which some processing module is waiting has been freed. The head of the chain of function blocks queued for a shared resource and the pointer to the chain of all function blocks are both maintained in the CR. Pointers to the Data Control Blocks (DCB's) for the TMS Library, Snap, and Log Data Sets are kept in the CR. Finally, several bytes of flags and a Program Interruption Element (PIE) save area are maintained in the CR.

1.2 Function Block

The Function Block (FB) is the major system table associated with a particular terminal device. Another approach is to consider each FB to be associated with a particular invocation of an application program (either separate, or sharing the program with other FB's). In this context, the top-level supervisor program may be considered to be a sort of super-program, and any application program a sub-program of it. The FB maintains most of the information unique to a particular terminal device and the program that is controlling it currently. In addition, it maintains direct pointers to those tables used most often in communication between the application program and TMS, OS/360, and the communication hardware.

A major part of the FB is a 16-word save area used to save the contents of all general registers when the FB does not have control of the CPU (i.e., is in TMS WAIT status). Individual byte fields are reserved for general FB flags, last-entry information, and FB queue flags. Pointers are maintained to the CR, the next FB on the chain of all FB's, the next FB on the chain of FB's for the same communication line, the next FB on the chain of queued FB's, the start of the application program's storage block chain, the start of the application program's Data Control Block Chain, the Data Event Control Block for the associated communications line, and the attached communication buffer.

Many communications-related items are kept in the FB. These include an Event Control Block for terminal I/O, the terminal list offset used to locate the entry for the associated hardware terminal, a copy of the polling/addressing characters for a multidrop terminal, the BTAM relative line number, the terminal number in EBCDIC, the current operation code for a terminal read/write operation, and various line length and position counts. Finally, the 4-character user name and the name of the current application program are kept in the FB for accounting and control purposes.

1.3 Wait List and Wait List Extension

The wait list forms the hub of TMS operation and is the reason that TMS is able to multiprogram in a single partition. The main portion of the wait list is exactly what the name implies: a standard OS/360 multiple-event wait list. The wait list extension is the same length as, and immediately follows, the last position of the wait list itself. Each word of the wait list extension is associated with the corresponding word in the wait list and can be located by adding the wait list offset from the CR to the address of the wait list entry. The wait list and wait list extension both consist of a static (or fixed-length) component followed by a dynamic (or variable-length) component.

Three distinct sections which must be recognized by the TMSWAIT routine actually comprise the wait list. The first section consists of points to specialized TMS Event Control Blocks. First

comes an Event Control Block pointer for input from the operator's console, followed by a pointer to the queuing Event Control Block located in the CR. The corresponding entries in the wait list extension are not used. The second section consists of a pointer to each communication line Event Control Block. To indicate that these are line Event Control Blocks, the first byte of the corresponding word in the wait list extension is set to hexadecimal 'FF'. These first two sections comprise the fixed-length component of the wait list. The third section consists of a pointer to an Event Control Block for each FB which is currently in wait status. This section varies in length as FB's are added to or dropped from the list, so it is being reordered constantly. The corresponding entries in the wait list extension point to the associated FB.

1.4 Teleprocessing Data Event Control Block

The Teleprocessing Data Event Control Block (TDECB) is the major system table associated with a particular communication channel. As such, it can be shared by more than one communication terminal, and thus may be pointed to, and shared by, several FB's. The TDECB is the principal table used by OS/360 BTAM (Basic Teleprocessing Access Method) to control all data communications activities. The first word of each TDECB, sometimes called the line ECB, is pointed to by the wait list. This is how TMS knows when a communication operation has completed.

The first 40 bytes of the TDECB are a standard IBM OS/360 BTAM DECB. The contents and uses of the fields therein are documented in the IBM publications regarding BTAM, so they will not be described here. The remainder of the TDECB consists of fields specific to TMS. There are several pointers: one to the head of the chain of FB's associated with this TDECB; three to the various code translation tables for the particular class of communication devices served; and one to the terminal list, which is a list of entries giving the polling characters and control flags for the several terminal devices associated with this TDECB. Several bytes and halfwords are used for both bit flags and counts that describe the type of communication devices being serviced; some of the devices' hardware parameters such as character capacity; and the current state of the communication channel and the terminals attached to it. Finally, a save area for some fields is provided for use when read polling must be interrupted for a write operation and then must be resumed.

2. INTERNAL SYSTEM CONVENTIONS

This section describes the various conventions that all TMS modules and all application programs must follow if TMS is to function properly. Each subsection describes a convention or several related conventions regarding a particular TMS function.

2.1 Save Areas

To the application programmer, the rules for using save areas are the same as for OS/360, except that the contents of the first word of the existing save area must be copied into the first word of a newly-obtained save area. This insures that Register 13 will always point to a word containing the address of the relevant FB.

To avoid confusion, each FB must have a separate chain of save areas associated with it. The first save area on this chain is obtained and formatted by the "phantom job" when it is entered. Since the phantom job is entered once for each FB and is completely re-entrant, the independence of the various save area chains is assured.

Most TMS functions involve receiving a request from an application program, editing and checking the request for validity, and then requesting OS/360 to perform certain tasks in order to satisfy the original request. This approach requires the use of two save areas: one to save application program registers upon entry to TMS, and one to save TMS registers upon entry to OS/360. One of these save areas is provided by the save area chain associated with the FB (i.e., the save area pointed to by register 13); the other save area is the first 16 words of the FB itself. In practice, a TMS processing module that is entered by an application program first saves registers in the usual manner while it locates the FB. It then moves the saved information into the FB save area and stores the contents of Register 13 at the end. The save area pointed to by register 13 is then free for use by OS/360.

2.2 CR and FB Pointers

As the two crucial blocks involved in every TMS activity, the CR and FB must be reachable in many ways. The CR and FB may be located under any of the following conditions:

a. The TMS execution-time monitor TMSEXEC is first entered from OS/360. The address of the CR has been placed in register 1 (parameter register) by the TMS startup routine TMSHSKP.

b. A TMS processing module is called from an application program. Register 13 points to a save area, the first word of which contains the FB address. The FB contains a pointer to the CR.

c. A module in TMSEXEC must operate upon all FB's. The CR address is obtained from the entry point TMCRADR in TMSBEGIN. The chain of all FB's is found from CRFBCHN, and each FB is operated upon in turn.

d. The communications transmission end routine TMSTREND is entered from TMSWAIT and must locate the FB corresponding to the terminal just contacted. The address of the TDECB block is known upon entry to TMSTREND. This block has a pointer to the chain of all FB's associated with that particular line. Each FB in this chain is checked until the one matching the last active entry in the BTAM terminal list is found.

e. The Event Control Block (ECB) that an application program has been waiting on is posted complete, and TMSWAIT must return control of the machine to that program. When TMSWAIT locates the address of the proper ECB in the wait list, the corresponding word in the wait list extension points to the associated FB.

f. A shared resource for which one or more FB's are waiting has just become free, and the next FB to use it must be found. The CR address is known, and the CR points to a chain of queued FB's. The dequeuing routine travels down this chain of FB's until it locates the first FB whose queue flags indicate a need for the resource in question.

2.3 Chain of Core Storage Blocks

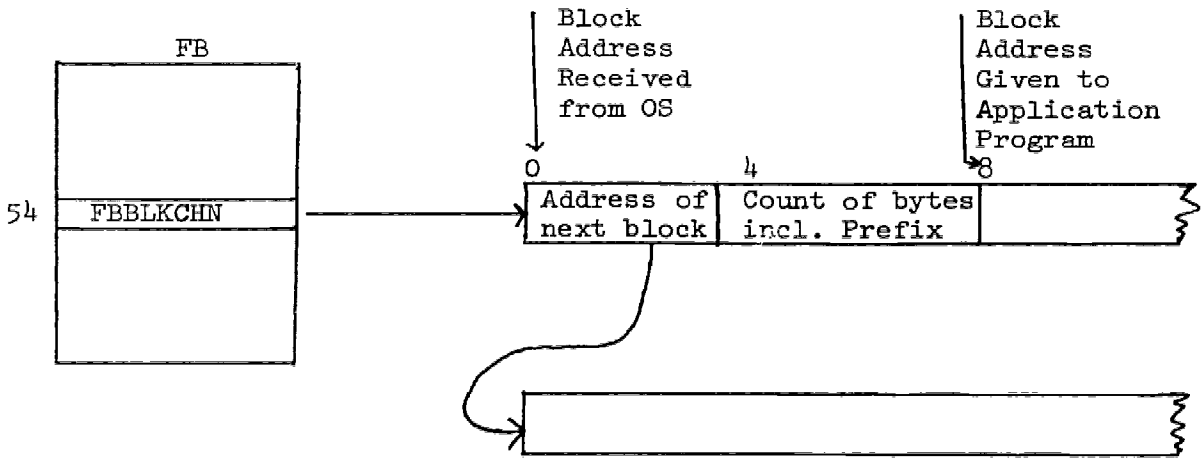
In order for TMS to "clean up" after an application program has gotten into trouble, it must be able to free all main storage obtained by that particular invocation of the application program. This process requires that the size and location of every block of main storage for a particular FB be known to the system. This is accomplished by a system of chaining together all blocks of storage.

All storage that is requested directly by an application program is obtained through TMS. The system adds 8 bytes to the request as received from the application program and passes it on to OS/360. When the request is satisfied, TMS uses the first 8 bytes of the storage block as a special chaining prefix as indicated in Fig. 1. When an application program requests the freeing of main storage, TMS checks the chain to verify that the storage is actually associated with the program.

2.4 Chain of Data Control Blocks

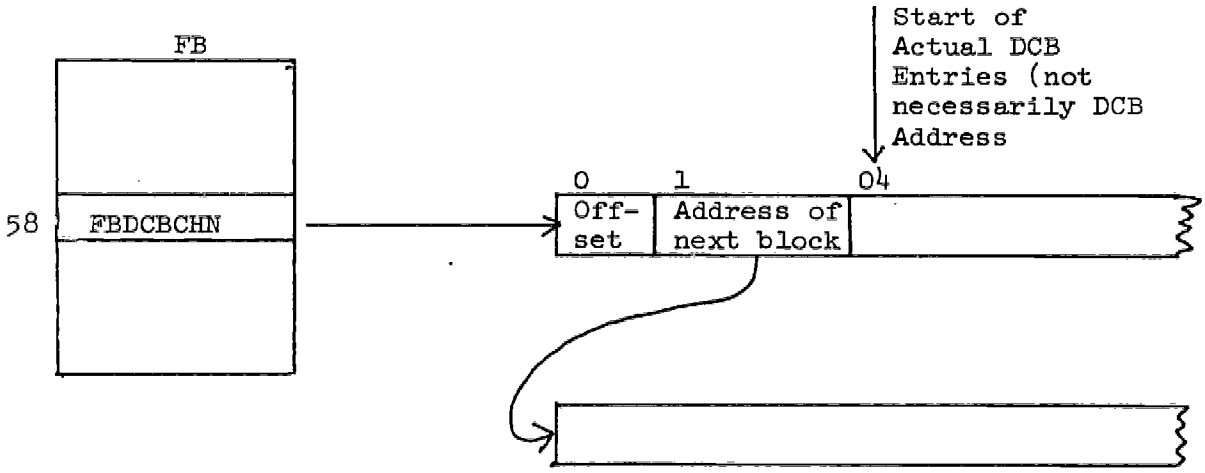
For a particular invocation of an application program that gets into trouble, TMS must have a method for closing all Data Control Blocks (DCB's) for the same reasons that require the ability to free all main storage blocks. The solution of this problem is quite similar to that employed for the main storage problem.

FIG. 1
STORAGE BLOCK CHAINING



* * * *

FIG. 2
DATA CONTROL BLOCK CHAINING



Every DCB for a data set that is opened by an application program is constructed by TMS in main storage freshly obtained for that purpose. Prior to the first byte of usable DCB information, TMS inserts a 4-byte prefix pointing to the next DCB for that FB and providing an offset to generate the true DCB address. The actual arrangement is shown in Figure B. The offset is a value which is subtracted from the address of the first byte following the 4-byte prefix to give the actual DCB address for use by OS/360. The length of the DCB is implied by bits within the DCB which describe the type of data set that it represents.

2.5 Types of I/O and Data Sets Supported

The Terminal Monitor System supports a subset of all of the I/O options and data set type supported by OS/360. This subset was chosen to give the broadest possible range of data set handling capabilities consistent with the basic objectives of TMS. For example, features unique to certain peripheral devices such as card readers, printers, and magnetic tapes were omitted since TMS is designed to deal with only direct storage device and communication terminals. The requirement that TMS retain all control over WAIT scheduling, as well as the stringent limitations on overall systems size dictate that TMS limit itself to only the basic (as opposed to the queued) access method. In general, these restrictions are enforced by code in the TMSOPEN macro which refuses to recognize any options not allowed by TMS.

Although it is not stated in any documentation available to the casual user, TMS does support the EXCP access method with or without appendixes. This access method, thus, is available for use by highly skilled personnel in implementing specialized or unique functions under TMS. The basic sequential access method (BSAM) is available along with CONTROL and NOTE/POINT features. The CONTROL feature is intended only for use in file positioning, and not for such specialized functions as line skipping on a printer or stacker selection in a card reader or card punch. The options permitted under BSAM include the control of data checks for a printer with a universal character set feature and the WRITE validity check. The basic direct access method (BDAM) is supported for searching by block ID and searching by key. The CHECK specification is also permitted. Optional services supported under BDAM include extended search, write validity check, feedback control, and the use of actual device addresses or relative block addresses in place of the standard TTR addressing scheme. The basic index sequential access method (BISAM) is supported only for reading and updating; the use of CHECK is also supported. No BISAM options are supported under TMS. Finally, the basic partitioned access method (BPAM) is supported for basic reads and write. The only BPAM option supported is a write validity check.

As the preceding indicates, TMS supports the four basic types of data set organization: sequential, partitioned, index sequential, and direct. The sequential partition and direct data sets may be either movable or unmovable. The types of I/O

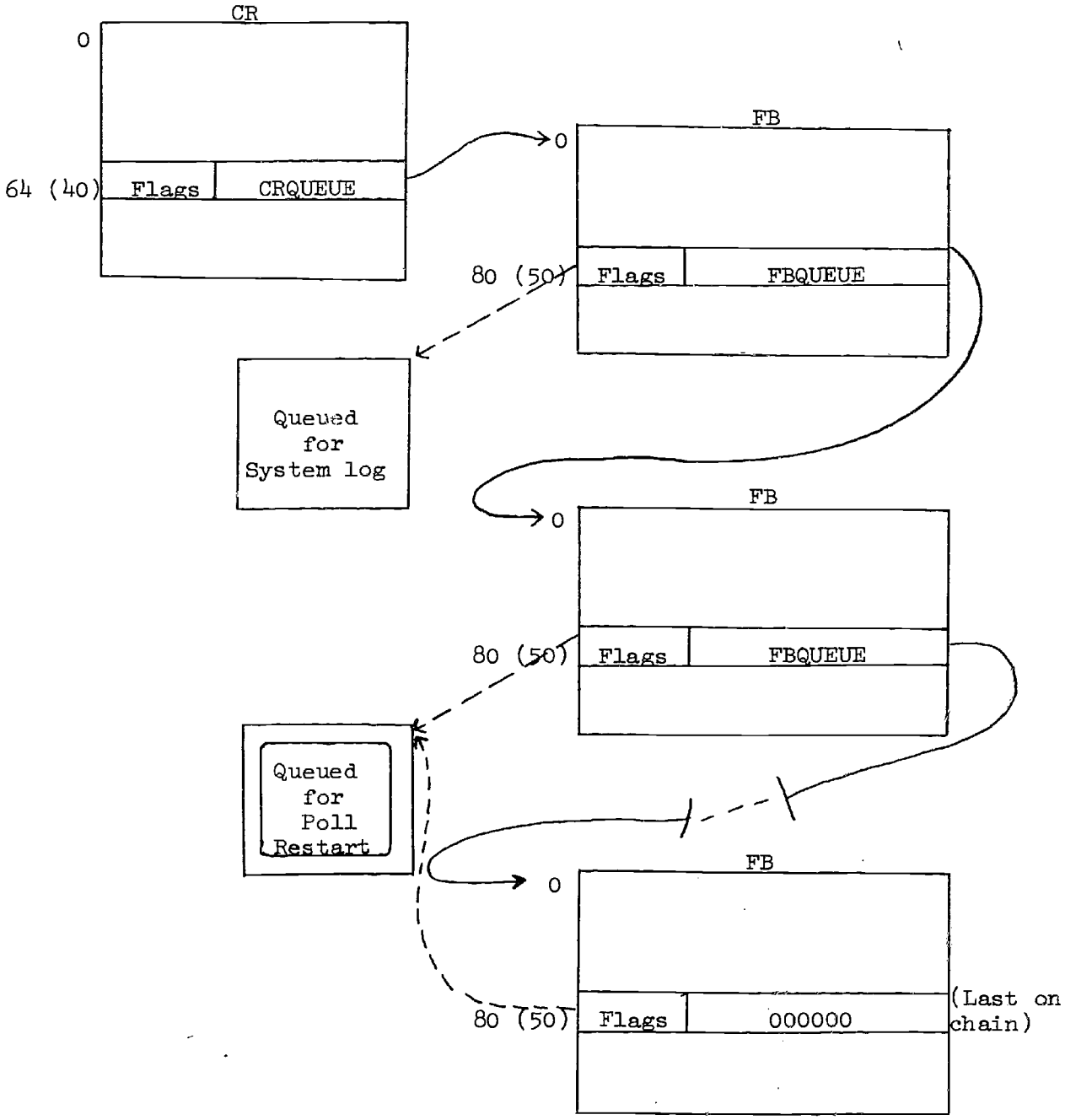
processing support include INPUT, OUTPUT, INOUT, OUTIN, and UPDAT. These parameters have the same meaning as in the OS OPEN macro instruction.

2.6 Queuing and Dequeuing

The terminal monitor system contains some resources which are shared by the application programs, but which may not be used by two or more programs simultaneously. The official IBM terminology for such a resource is "serially reusable." Rather than use the relatively complex ENQUE/DEQUE facility of OS/360, TMS incorporates its own fairly simple queuing and dequeuing facility. Enqueuing is primarily a function of the processing module which represents or controls the serially reusable resource in question. Dequeuing is primarily the responsibility of the TMSWAIT processing module. Two words in the CR and one word in each FB are used for queuing control. The various processing modules use the CR dummy ECB to indicate when a shared resource which is being waited for has become free. The particular shared resource, which is now free, is indicated by setting a bit, in addition to the completion bit. The CRQUEUE pointer points to the chains of queued FB's. The first byte of this word, also known as the CR queue flag area, has a bit set on for every shared resource that is currently being waited for. In this respect, it is the logical OR of all the analogous fields in all FB's. The FB queue pointer is used to point to the next FB on the queue. The first byte of this word, also known as the FB queue flag, will have only one bit set on (in addition to the end-of-queue bit, if necessary) which will indicate the particular resource for which the FB is waiting. These relationships are illustrated in Fig. 3.

Placing an FB on the queue involves appending the FB to the end of the queue chain, setting the proper bit in the FB queue flag area, and ORing this FB queue flag area with the CR queue flag area. When the processing module which is releasing a shared resource wishes to detect whether another FB requires this resource, it merely checks the CR queue flag area. The assignment of a new FB to the shared resource is accomplished by searching down the chain of queued FB's until the first queued FB with the proper bit set on is located. This FB is then removed from the chain, but the bit representing the shared resource is not reset in the CR queue flag area. Since more than one FB may have been waiting for the resource, this bit is reset only when a search of the chain has failed to turn up any FB with the corresponding bit sets.

FIG. 3
FB QUEUING



3. INTRODUCTION TO MODULE FUNCTIONS

3.1 System Initialization

System initialization consists of setting up all the specialized TMS system tables, opening certain system data sets, and loading several programs, principally IBM access method modules into upper core. With initialization completed, control is passed to the processing routines in the execution time load module. Three processing modules are involved in system initialization. Two of these, TMSHSKP and TMSBLOCK, comprise the system initialization load module. The third module, TMSBEGIN, is a small processing module in the execution time load module, and it provides the main entry point to the execution time load module. TMSHSKP contains a great bulk of the code for system initialization. It performs all initialization operations except those that require information available only within the execution time load module, such as the entry point addresses of the various execution time processing routines. TMSBLOCK is the basic skeleton for most of the TMS tables. It is a separate processing module, so that any changes in the configurations of communication devices serviced by TMS may be accomplished merely by reassembling TMSBLOCK. Among the tables for which skeletons are provided in TMSBLOCK are FB's, TDECB's, DCB's for all communication lines and system data sets, all communication buffers, all terminal lists, and all translation tables. TMSBLOCK has three entry points: the entry point TMSBLOCK, which points to the first word of the skeleton; the entry point TMSBLGTH, which is a word containing the length of the skeleton in bytes; and the entry point TMSLSTFB, which points to the start of the last FB skeleton assembled which will be the head of the FB chain.

The TMSBEGIN module performs the last few steps associated with system initialization and then turns control over to the TMSWAIT module to begin executing each FB in turn. It too has three entry points: TMSBEGIN, which is the entry point to the final initialization code and by extension to the entire time load module; TMSCRADR, which is single word containing a pointer to the CR for use by any other execution time processing module that needs it; and TMSSYSV, which is a standard 18 word save area used to save the register contents upon entry to the execution time load module.

3.2 Wait Handling and Dequeuing

All wait list handling and scheduling of the next FB to gain control of the CPU is handled by the processing module, TMSWAIT. It is intended to be the only module that ever relinquished its control of the CPU to a lower priority partition. Since the TMS dequeuing method employs a pseudo-ECB, TMSWAIT is responsible also for locating the next FB that requires the shared resource and for passing control to the relevant processing module. TMSWAIT has two entry points: the entry point TMSWAIT, which is used for

entry from application programs; and the entry point TMSDWAIT, which is used for a direct entry to the wait module from other processing modules. The two separate entry points are necessary because direct entry into the wait module does not involve saving register in an FB.

3.3 Communications with Computer Operator

The TMSCNSL is responsible for the receipt and execution of all commands from the central computer operator and the reissuing of the WTOR to reset TMS for the receipt of the next command. Any other processing module may transmit a message to the computer operator, as long as it does not wait for a response. The TMSCNSL routine has two entry points: TMSCNSL, which is entered from TMSWAIT when the console ECB indicates that a message has been received from the central computer operator; and TMSREADY, which is entered from TMSBEGIN to complete system initialization by issuing the first WTOR of the TMS run.

3.4 Communications with Terminals

All transmission of messages to and from remote terminals is handled by two processing routines. The first, TMSCISO, is used for the initiation of all terminal input/output operations. The second, TMSTREND, is entered from TMSWAIT when OS/360 indicates in one of the line ECB's that some terminal I/O operation has been completed. This routine performs all operations associated with the completion of terminal I/O. IMSCISO performs such operations as determining whether or not the communication line is currently free, queuing the operation if it is not; selecting the buffer; performing code translation for outgoing messages; setting up the TDECB for the transmission; and invoking OS/360 BTAM to initiate the actual I/O operation. It has four entry points: TMSCISO, which is entered from an application program or another processing module to initiate reading or writing; TMSWRDEQ, which is entered from TMSWAIT to initiate writing by an FB just removed from the FB queue, where it had been placed until the line became available for a write operation; TMSRDRST, which is entered from TMSWAIT to restart a read polling operation on a multi-terminal line interrupted for a write transmission to one of the terminals; and TMSCSIOR, which attempts recovery from certain types of transmission errors.

The TMSTREND routine performs such operations as checking for transmission errors, performing code translation on incoming messages, and determining which terminal originated an incoming message so that it may post the I/O operation complete in the ECB for the proper FB. It has one entry point, TMSTREND.

3.5 Obtaining and Releasing Prime Storage

All obtaining and releasing of prime storage for application programs is handled by the TMSGMFM processing routine. This rou-

tine maintains the system convention regarding the chain of core storage blocks for each FB and performs validity checking against this chain. It also preserves the integrity of TMS by converting an application program's unconditional request for primary storage to a conditional request for primary storage on behalf of the monitor system. Thus, it retains control of the situation if the request for storage cannot be honored by OS/360. This routine has a single entry point, TMSGMSM; the type of operation desired is determined by its parameters.

3.6 Locating, Opening, and Closing of Data Sets

There are two classes of data sets in TMS: system data sets and application program data sets. System data sets include the TMS library, snap and log data sets, and all communication lines. By their very nature, their requirements are known in advance, and they are opened by the TMSHSKP routine. The remainder of the data sets are opened and closed in response to requests from application programs. Since the opening and closing of data sets offer many opportunities for system crashes, all of this activity is performed for application programs by two processing routines: TMSOPEN and TMSCLOSE.

TMSOPEN performs such functions as checking that a data set of the proper name exists and that all volumes containing it are available to the system; checking that all volumes listed as containing the data set have a proper entry for that data set; obtaining primary storage for and constructing a DCB for that data set; maintaining the system convention of a chain of DCB's for the application program's FB; forming a JFCB for the data set (since the TMS deck does not include a DD card for every data set); obtaining primary storage for a buffer if necessary; and issuing the OS/360 OPEN macro for the data set and verifying the successful completion of the open process. This routine has one entry point, TMSOPEN.

The TMSCLOSE routine performs such functions as: checking request validity; issuing the OS/360 CLOSE macro; freeing any buffer space obtained by TMSOPEN; and removing the DCB from the DCB chain and releasing the primary storage that had been obtained for it. It has two entry points: TMSCLOSE, which is entered from an application program; and TMSPCLOS, which is entered from the TMSPURGE routine to close data set with a slightly different parameter structure and no validity checking.

3.7 Loading Requested Programs

Since the loading of user requested programs can result in system aborts if (1) the program requested is not present or (2) insufficient primary storage remains to accommodate the program within the TMS partition, the processing routine TMSPLOAD is used. TMSPLOAD performs the following functions: executing an OS/360 BLDL macro to insure that the desired load module exists and to obtain its length

in bytes; searching the FB chain for an already loaded copy of the same program which is re-entrant (thus insuring that additional primary storage will not be required); testing to insure that there will be sufficient primary storage for both the program to be loaded (if necessary) and the load routine; and invoking the OS/360 LOAD macro instruction to complete the loading process. This routine has one entry point, TMSPLOAD.

3.8 Recovery from User Program Errors

When a running application program has caused an error condition which is detected by one of the TMS processing modules, and the application programmer himself does not provide for recovery from this error situation, TMS must terminate the application program so that all system resources used by that application program again are made available for the remainder of the system. It is also highly desirable that the terminal user be notified in a uniform manner of the various types of errors. All these functions are performed by the TMSPURGE processing routine. This routine specifically performs the following functions: the closing and fraying of all attached DCB's; the releasing of all primary storage obtained by the application programs; the locating of the topmost save area on the save area chain; the issuing of a suitable error message selected from a table of error messages by a parameter to the purge routine; and the returning to the phantom job module (TMSPJOB) to indicate an abnormal exit from the application program. It has a single entry point, TMSPURGE.

3.9 Top Level Control

In a system where there is a choice of many different application programs, there must be a top level control program which handles the initial dialog between the terminal user and the system. Such a program invokes the various programs in response to the user's request and handles the transitions between not only application programs but also successive users at the same terminal. To maintain simplicity and flexibility, this particular program is not considered a processing module like the remainder of TMS, but rather is treated as just another application program. The only difference between this and other application programs is that it has been loaded as part of the system initialization process, and all the FB save areas have been preset to return to the entry point of this program as if it has just called TMSWAIT to wait on an event which is now complete. One of the names that has been coined for this pseudo application program is "phantom job," and from this the program module takes its name, TMSPJOB. Some of the functions performed by TMSPJOB are: issuing the initial sign-on message to each terminal when TMS begins operation; requesting, receiving, and verifying the user's identification code as part of the job in process; requesting and receiving the name of the application program that the user desires to employ; calling TMSPLOAD to bring the program into primary storage; initializing the FB for use of the TMSDEBUG routine if the user so desires; passing control to the requested application program and

eventually receiving control back from the program by either a normal or an abnormal return; logically disconnecting the terminal from the system at the user's request; and accepting the user's log-out when he is finished and conditioning the system to accept the log-in of a new user from the same terminal. This routine has one entry point, TMSPJOB.

4. DETAILED MODULE DESCRIPTIONS

4.1 TMSBEGIN Module

The TMSBEGIN module has three entry points. The main entry point is TMSBEGIN, which is used to initiate TMS execution time processing and which becomes the main entry point of the load module TMSEXEC. The entry point TMSCRADR obtains the communication region address if it cannot be located by any other method. The entry point TMSSYSSV is used to gain access to the TMS system save area if other modules need to reach it.

This module customarily receives control by being transferred to and from TMSHSKP by means of an XCTL macro instruction. As is customary, it does a standard OS SAVE macro instruction and links the main TMS system save area to the save area pointed to by the system. TMSEXEC has provided the pointer to the communication region (CR) in register 1 (RP1). This address is now moved to register 10 (RCR) and installed in the area pointed to by TMSCRADR.

A vector of entry point addresses to the various TMS processing modules is maintained at location ADDRVEC. This consists of a series of V-type address constants in the same order as defined in the CR dummy section (DSECT). The length of this address vector is found as the value of AVLENGTH. This address vector is moved to the first part of the communication region by means of a single MVC instruction.

Since certain serially reusable system resources may be requested by more than one applications program at the same time, a queuing methodology is supported. The key to signaling when a resource has been freed is the queuing event control block. The address of this event control block is placed in the second position of the wait list at this time. This procedure is explained in detail in the description of the TMSWAIT module.

The only function left in the monitor initialization process is the issuing of a ready message to the console typewriter by means of a WTOR macro instruction, so that the operator may exercise control over the system from his console typewriter. Since this process is one that is repeated every time the operator employs the console typewriter, a branch is taken to the special entry point TMSREADY in the TMSCNSL module.

4.2 TMSBLOCK Module

The purpose of the TMSBLOCK module is to assemble itself into the terminal block skeletons and associated pointers for use by the TMSHSKP module. This module has three entry points: the entry point TMSBLOCK represents the beginning of the combined terminal block skeletons; the entry point TMSLSTFB provides the address of

the last FB generated in the TMSBLOCK module; and the global character variable &PREVFB is used to obtain the name of the last FB generated by the FORMFB macro.

The principal portion of terminal block skeleton generation is performed by repeated calls of the FORMFB macro. Details on this macro's parameters and the code that it generates may be obtained from the section entitled "Use of the FORMFB Macro." All invocations of FORMFB should be placed between the headings "START OF MACRO CALLS TO DEFINE TMS BLOCKS" and "END OF MACRO CALLS TO DEFINE TMS BLOCKS" with no other intervening macro instructions, machine operations, or assembler pseudo-instructions which alter the location counter placed within this area. Following the area in which TMS blocks are defined come the various translation tables which are needed to convert code from the internal EBCDIC to the various transmission codes employed by the different communications devices. Each table is 256 bytes long and is labeled with the characters IECT followed by the four characters used in the INTRAN, OUTTRAN, or INTRLC keyword operands of the relevant FORMFB macro instruction.

4.3 TMSCLOSE Module

TMSCLOSE is a service module of the Terminal Monitor System designed to close down user files. It begins with a standard store multiple of the user's registers into the save area he has provided. The function block of the user is located, and the register is copied, from his save area into location FBSAVE in the FB. The address of the data control block (DCB) is copied from register 1 into register RDCB; register RDCB is then used to provide addressability for the dsect IHADCB. The chain of DCB's is then checked to make sure that this DCB address is on that chain. Location FBDCBCHN in the FB is checked for zeroes. If it is all zeroes, then the chain is empty, and a branch is taken to location DISASTER. The chain, if not empty, indicates the presence of DCB's. In this case, the address of the DCB chain is loaded into register RPREV at location TESTDCB1, register RWORK is cleared, and the control word for the next DCB is loaded into register RWORK2 from the location currently pointed to by register RPREV. Register RWORK2 now points to the control word in front of the next chained DCB. (See description of TMSOPEN for construction of DCB.) The negative offset at this address is loaded into register RWORK1 and used to compute the true address of the start of the DCB. This address is then compared to the address given on entry to the program. If equal, this is the DCB the user wished to close, and control is passed to location TESTDCB2. Otherwise, register RPREV is loaded from the address to which it currently points. This step will give the address of the next DCB. This address is checked for zeroes. Zeroes here indicate that there are no more DCB's chained to this FB. If this is the case (an incorrect DCB address passed to TMSCLOSE), a branch is taken to location DISASTER. Otherwise,

since the address of the next chained DCB is in register RPREV, a branch is taken back to location TESTDCB1 to test the next DCB. At location TESTDCB2 the location DCBMACRF in the DCB is tested for indications of a QISAM DCB. If this was a QISAM DCB, the module TMSGTSLE must be deleted. If this was not, a branch is taken to location CLOSE.

At location CLOSE an O/S CLOSE macro instruction is issued, specifying the location of the DCB. The next task after closing the file is to free the core obtained for this DCB. To do this, location DCBBUFCB is tested for the presence of a hexadecimal 01. If it is present, it implies the existence of a buffer pool connected to this DCB which also must be freed along with the core of the DCB itself. If it is not present, a branch is taken to location FREEDCB. Assuming there is a buffer pool connected to this DCB, the pointer to this buffer pool is picked up from location DCBBUFCB and placed in register RPTR. The number of buffers in the buffer pool is picked up from four bytes off the address currently in register RPTR and placed into register RCTR. The number of buffers (now in register RCTR) is multiplied by the buffer size which is found at six bytes off the address currently in register RPTR. The alignment of the buffers is tested by testing location DCBBFALN for the presence of a hexadecimal 01, which indicates fullword instead of doubleword alignment. If location DCBBFALN does not contain a hexadecimal 01, there is doubleword alignment, and a branch is taken to location BUFFER1. If buffer alignment is on a fullword, register RCTR is incremented by 4, and control then falls through to location BUFFER1. At location BUFFER1 the length of the buffer pool currently in register RCTR is incremented by 8 to account for the buffer control block. This length is placed into register RP0, and the address of the buffer control block currently in register RPTR is placed into register RP1, and an O/S FREEMAIN macro instruction is issued to free the core for the buffer pool. Control then falls through to location FREEDCB.

Once the core for the buffer control block and the buffer pool have been released, the task is to free the core for the data control blocks themselves. Since the length of a data control block is dependent upon the access method, location DCBMACRF must be tested to find the access method used. At location FREEDCB in the program, location DCBMACRF in the user's DCB is tested for the presence of a bit indicating the EXCP access method (DCBBITE). If this bit is not present, a branch is taken to location FREEDCB1. Assuming this access method is EXCP, register WORK1 is loaded with a value of 56. (This is 52 bytes for the EXCP DCB plus four bytes for the chain word.) Location DCBMACRF is then tested for the presence of a bit indicating that appendages were needed for this access method. If this bit is not present, a branch is taken to location FREEDCB5. If it is present, register WORK1 is incremented by 20 bytes, and a branch is taken to location FREEDCB5.

At location FREEDCB1, location DCBDSORG is tested for the presence of a bit indicating an index sequential data set (DCBBITIS). If this bit is not present, a branch is taken to FREEDCB2. Assuming the bit is present, register RWORK1 is loaded with a value of 244 for the 236 bytes of the ISAM DCB, plus four bytes for the chainword, plus four bytes used by TMSGTSLE. A branch is then taken to location FREEDCB5.

At location FREEDCB2 location DCBDSORG is tested for the presence of a bit indicating the direct access method (DCBBITDA). If this bit is not present, a branch is taken to location FREEDCB3. Otherwise, register RWORK1 is loaded with a value of 92 for the 88 bytes for the direct access data control block plus 4 bytes for the chainword. A branch is then taken to location FREEDCB5.

At location FREEDCB3 location DCBDSORG is masked for the presence of a bit indicating the physical sequential access method (DCBBITPS). If this bit is not present, location DCBDSORG does not contain a valid bit representation of the access method. A branch is then taken to location DISASTER. If this is a physical sequential data set, register RWORK1 is loaded with a value of 92 for the 88 bytes for the data control block plus the 4 bytes for the chainword. At location FREEDCB5 the DCB chain is updated by moving 3 bytes from the area pointed to by register RBLOCK (which contains the address of the data control block immediately following the one just closed) and moved to the address contained in register RPREV. This action completes the chaining of the previous data control block to the succeeding data control block. Register RP0 is loaded from register RWORK1 which contains the size of the data control block area to be freed. Register RP1 is loaded from register RBLOCK which contains the address of the area to be freed. Then an O/S FREEMAIN macro instruction is issued.

Control then falls through to location RETURN where a standard load multiple of the user's registers from location FBSAVE occurs, and a branch register return through register RR. At location DISASTER, register RP1 is loaded with a value of 28 to indicate entry from TMSCLOSE, and register RD is loaded with a V-type address constant of the entry point of TMSPURGE. A BR instruction is issued specifying register RC.

Location TMSPCLOS is an alternate entry point to TMSCLOSE. This is an entry from TMSPURGE. On entry, register RP1 points to the chain element word rather than to the DCB. At location TMSPCLOS, the registers are stored into location FBSAVE. Register 15 is used for temporary addressability of location TMSPCLOS which enables an address constant specifying the entry point TMSCLOSE to be loaded into register RP. This provides permanent addressability for the program. Register RWORK1 is then cleared. The negative offset from the chainword is then inserted into register RWORK1. From this the address of the DCB may be found easily, and control is passed to location TESTDCB2.

4.4 TMSCNSL Module

The TMSCNSL module has two entry points. The principal entry point, TMSCNSL, is entered from the TMSWAIT module when the console DCB indicates that a message has been received from the computer operator. The message test which has been enclosed between apostrophes in the operator's response via the REPLY command is found in the internal buffer named RPLAREA. To test for the content of the message, a simple series of CLC instructions is used to compare the contents of the reply buffer to see if it matches a particular key word. If the first word of the response does not match any of the valid command words, control falls through to the routine labeled WHAT, which issues the message

TMS101I COMMAND NOT RECOGNIZED. IGNORED.

and then falls through to that portion of code headed by the label TMSREADY.

If the response from the operator matches the key word "DUMP," a user ABEND with a code of 300 is executed in order to obtain a core dump and terminate operation of the monitor.

The remainder of the module consists of code to condition the monitor system for the next response from the computer operator. This code is headed by the label TMSREADY, which is also the secondary entry point to this module. The code zeroes out the first byte of the console DCB area RPLECB and sets the response buffer RPLAREA to all blanks. The WTOR macro instruction is then invoked to write the message

TMS100I READY

on the console typewriter and direct OS to place the operator's response into the buffer labeled RPLAREA. The final operation is to place the address of RPLECB into the first position of the wait list. This is done every time, even though it is necessary that this be done only on the first call of TMSREADY from TMSBEGIN in order to complete the system initialization. The only exit from this module is direct branch to the direct wait entry point TMSDWAIT in the TMSWAIT module.

4.5 TMSCSIO Module

The TMSCSIO module has four entry points: TMSCSIO, TMSWRDEQ, TMSRDRST, TMSCSIOR. The main entry point to the routine is TMSCSIO. The entry point begins with a standard store multiple into the user's save area, locating the function block, moving the stored registers to location FBSAVE, and establishing addressability to the program. At this entry point also the flags are set to indicate a normal exit. Addressability to the communication region is provided by

loading register RCR from location FBCR. Loading register RDECB from location FBDECB in the function block provides addressability to the data event control block. Loading register RDCB from location DECDCBAD in the data event control block provides addressability to the data control block. The function block entry flag is then set to indicate entry into TMSCSIO. After the entry, the next task is to find whether a buffer is attached to this function block by checking location FBBUFPTR for the presence of zeroes. If there is no buffer attached to this function block, a branch is taken to location IOENTRY4. If there is, register RBUF is loaded with the address in location FBBUFPTR. Location FBBUFPTR is then cleared to zero, and the buffer is indicated to be the last by setting a flag (BUFFLAST). Register RPTR is loaded with the address in register RBUF offset +4 to point to the buffer itself rather than to the buffer control block. The length of the last message sent or received is loaded into register RPTR, and this length plus 4 (if it's a shared line) is used to clear this buffer to blanks. The clearing operation takes place in the loop at locations IOENTRY1 through IOENTRY3. At location IOENTRY4 the task is to test whether the request is a read or write operation. This is done by testing location FBRWOP for the presence of a flag indicating the write operation (FBRWRITE). If this flag is present, a branch is taken to location WRITE. If it is not, the read operation is resumed and a branch is taken to location READ.

At location WRITE the return address from the write queuing subroutine is set into register RR, and location DECUFLGS is tested for a bit indicating that writing is in progress (DECUFWIP). If writing is in progress, a branch is taken to location WRQUEUE to place this function block on the writing queue. If writing is not in progress, location DECUFLGS is tested for the presence of a bit indicating read polling in progress (DECUFRIP). If read polling is not in progress, a branch is taken to location WRITE8. If polling is in progress, it must be stopped for the other terminals on the same line so that this message may be sent. Register RR is loaded with the address WRITE0 to indicate a different return from the write queuing subroutine, and a branch is taken to the write queuing subroutine at location WRQUEUE.

At location WRQUEUE the queuing flags in the communication regions at location CRQFLGS are tested to see if the queue is empty (CRQEND). If it is not empty, a branch is taken to location QUEUE1. If it is empty, the address of the current function block is stored at location CRQUEUE. Location CRQFLAGS is set with a bit indicating that there is a function block queued for the write operation, and a branch is taken to location QUEUE4. If the queue was not empty at location QUEUE1, location CRQFLAGS is set with a bit indicating a function block queued for write (CRQWRITE), and the address of the first queued function block is loaded from location CRQUEUE into register RWORK. At location QUEUE2 a loop is executed to find the end of the queue. When the end of the queue is found, a branch

is taken to location QUEUE3. At location QUEUE3 the end-of-queue flag for the function block already queued is turned off. The queuing flags of the former FB are merged with the address of the new last entry and stored in the pointer. The flags indicating end-of-queue and a function block queued for write (FBQEND and FBQWRITE, respectively) are moved into location FBQFLAGS. The waiting-to-write count at location DECWWCNT is incremented by one, the waiting-to-write bit (DECUFWTW) is set on in location DECUFLGS, and a branch is taken to the return address that was previously loaded in register RR. If writing was already in progress, this location is TMSDWAIT. If read polling was in progress, this location is WRITEO.

At location WRITEO, the polling reset flag (DECUFPRS) is turned on in location DECUFLGS. The address of the data event control block is loaded into parameter register 1, and a RESETPL macro instruction is issued to stop polling on the line. When control is returned from this macro, register RC is tested for a return code of 0 to indicate a successful polling halt. If the polling halt was not successful, a branch is taken to location ABEND. Otherwise, a branch is taken to location TMSDWAIT. If read polling was not in progress at the entry to this routine, a branch is taken to location WRITE8 where the terminal list entry for this console is loaded by obtaining the address of the terminal list from location DECTLIST, placing this address into register RWORK1, and adding the terminal list offset for this function block from location FBTLOFF to the address in register RWORK1. The address in register RWORK1 is now the address of the terminal list entry for this console. Using this address, the skip bit for this entry is turned off, and location FBRWOP is tested to see if the length for the message is already in register RP0. If it is, a branch is taken to location WRITE21. If it is not in register 0, it is obtained from the start of the text and put into register RLNTH, and a branch is taken to the common coding at location WRITE22. At location WRITE21, register RPTR is loaded with the message address from parameter register 1, and register RLNTH is loaded with the length from parameter register 0.

At location WRITE22 the process is to locate an output buffer. To do this, register RCTR is cleared to 0. The address of the buffer control block is put into register RBUF from location DCBBUFCB, and at location WRITE1 a loop is executed to chain through the buffer chain to find a buffer that is free. If no buffer is free, a branch is taken to location ABEND with a value of 41 at location CRABCODE. If a buffer is available, control falls through to location WRITE2 where flags are set in the buffer control block for that buffer to reserve the buffer. These flags indicate (1) buffer in use and (2) buffer waiting for output (BUFFINUS and BUFFWRTG, respectively). Register RBUF is shifted by four bytes to point to the start of the buffer itself rather than to the control block. Location DECTYPE is tested to see whether the terminal is a typewriter or not. If it is not a typewriter, a branch is taken to location WRITE3. If it is a typewriter, location FBRWOP is tested to see if carriage return before text was specified (FBRWCRBW). If not, a branch is taken to

location WRITE3. If it was specified, the length of the last message is loaded into register RWORK2, register RWORK1 is cleared, and a halfword of zeroes is stored into location FBLMLNTH, which is the last line length. The value currently in register RWORK2 is divided by 10 to find the inches of carriage travel. The prefix length is added to the value in register RWORK2 to find the total prefix length. Register RWORK1 is loaded with a buffer address from register RBUF. The length value in register RWORK2 is temporarily decremented by 1, and this value is used to move a carriage return prefix into the buffer by an execute instruction specifying a move instruction at location MOVECR. The value in register RWORK2 is reincremented by 1 and this value added to the cumulative length in register RCTR. The text of the message is to be moved into the buffer at location WRITE3 by first testing register RLNTH for the presence of a zero length. If the length of the text is zero, a branch is taken to location WRITE4. Register RLNTH is then tested for a maximum length value. If it is over this maximum length, a branch is taken to location PURGE1. The address of the buffer is then loaded into register RWORK1, and the length of the existing text from register RCTR is added to it. Also, the length value in register RLNTH of the existing message is added to the cumulative length counter register RCTR. The length of the outoging message is stored into location FBLMLNTH.

At location WRITE3O the length value in register RLNTH is used in a move loop to move the total message into the output buffer. When the message has been moved, control falls through to location WRITE4 where, if the terminal is a typewriter and a carriage return after text was specified, the same code that was issued for carriage return before text is executed. If the terminal is not a typewriter, or carriage return after text was not specified, a branch is taken to location WRITE5 where the type of terminal is tested again to see if the terminal is a display. If it is not, a branch is taken to location WRITE5A. If it is a terminal, the buffer address in register RBUF is loaded into register RWORK1 and the length of the existing text in register RCTR is added to it. This address is one beyond the end of the message, and if the terminal is a display, an end of text character is placed at that position, and the message length bumped by 1. At location WRITE5A, the cumulative length in register RCTR is put into register RLNTH, and the final buffer address is put into location RPTR. The translation table address is loaded into register RWORK1, and at location WRITE6 and WRITE7 a translate loop is executed to translate the outgoing message. The write-in-progress flag is then turned on at location DECUFLGS (DECUFWIP). The buffer length from register RLNTH is stored in the data event control block at location DECLNGTH. The address of the buffer is stored into location DECAREA. The polling address from location DECTLIST is loaded int register RWORK1, and the offset for this function block from location FBTLOFF is added to it. This address, pointing to the terminal list entry for this function block, is stored at location DECENTRY. The relative line number for this terminal is moved

from location FBRLN to location DECRLN, and if the terminal is a display, a branch is taken to location WRITE9. Otherwise, the type of operation in the data event control block is set to indicate a write initial with reset (DECWTIR) for a typewriter, and then the CRT display code is skipped by branching to location WRITE11.

At location WRITE9 the operation type at location FBRWOP is tested to see if pre-erase was specified. If it was, a branch is taken to location WRITE10. Otherwise, the type of operation at location DECTYPE+1 is set to indicate a write initial with reset (DECWTIR), and a branch is taken to location WRITE11. If pre-erase was specified at location WRITE10, the operation type at location DECTYPE+1 is set to indicate a write erase with reset (DECWTSR), and control falls through to location WRITE11. At location WRITE11 the data event control block is cleared to zero, and the write to the terminals is issued by issuing an O/S WRITE macro instruction. After control is returned from this macro instruction, register RC is tested for a return code of 0 to indicate a successful start of write. If it was not, a branch is taken to location ABEND with a value of 42 at location CRABCODE. If the start of write was successful, a branch is taken to the exit routine at location RETURN.

If the indicated operation at location IOENTRY4 was a read operation, a branch is taken to location READ. At location READ register RWORK1 is loaded with the address of the terminal list obtained from location DECTLIST in the data event control block. To this value is added the terminal offset found at location FBTLOFF. The skip bit at the resultant location is turned off, the type field of the data event control block (DECTTYPE) is tested for the presence of a bit indicating several terminals on the line, and a branch is taken to location READ0. If there are several terminals on this line, the active polling count at location DECAPCNT is incremented by 1.

The data event control block flags at location DEBUFLGS are tested for bits indicating either a read in progress or polling interrupt (DECUFRIP and DECUFPIN, respectively). If polling is or was in progress, a branch is taken to location RETURN to return to the caller, as nothing more needs to be done. If polling was not in progress, it must be initiated, and control falls through to location READ0 to locate an input buffer. The address of the buffer control block is obtained from the data control block, and the first buffer is tested for the presence of a flag indicating buffer in use (BUFFINUS). If the buffer is not in use, a branch is taken to location READ2. If it is in use, a loop is executed to chain down through the buffers to find a free one. If one can not be found, a branch is taken to location ABEND with a value of 43 at location CRABCODE for the abnormal end code in the communication region.

At location READ2 the flags on the newly-located buffer are set to indicate buffer in use and buffer waiting for input (BUFFINUS and BUFFWTIN, respectively.) Register RBUF is incremented by 4 to point to the start of the buffer. Next, the data event control block flags are tested for the presence of a bit indicating write in progress (DECUFWIP). If it is in progress, a branch is taken to location READ4 to put the read parameters into a save area. If it is not in progress, the operation code of a write initial (DECRTI) is moved into the type field of the event control block to set the operation code for this operation. At location READ21 the length field is set into the data event control block from location DCBBUFL. The buffer address now in register RBUF is stored into location DECAREA. The terminal list address is moved from location DECTLIST to location DECENTRY. The relative line number for this terminal is also moved from the function block at location FBRLN to the data event control block at location DECRLN. The data event control block flags are set to indicate read in progress. The data event control block is cleared, and an O/S READ macro instruction is issued. After the macro instruction is issued, polling is in progress, and a branch is taken to location RETURN.

If, at location READ2, the line is found not to be free, a branch is taken to location READ4 to save the current parameter set for a later restart of the read operation. At location READ4 location DECRSAVE+10 is tested for the presence of flags indicating that a restart parameter set already exists. If it does, there is a fatal error, and a branch is taken to location ABEND with a value of 45 at location CRABCODE. Otherwise, the first byte of the operation code, a hexadecimal zero, is moved to location DECRSAVE in the restart parameter save area. The second byte of the operation code, a read initial (DECRTI) is moved to location DECRSAVE+1. Likewise, the buffer length from DCBBUFL, the buffer address in register RBUF, the terminal polling list address from location DECTLIST+1, and the relative line number from location FBRLN are moved into respectively higher locations in the parameter save area. The data event control flags are set to indicate polling interrupted (DECUFPIN) and a branch taken to location RETURN to exit to the caller.

At location RETURN the program flags are tested for a bit indicating a direct entry to the wait routine is to be taken. If it is there, a branch is taken to location TMSDWAIT for a special exit. Otherwise, the function block entry flags are cleared, the user's registers from location FBSAVE are restored, and a branch taken back to the user routine. At location TMSDWAIT the register RC is loaded with a V-type address constant specifying the location TMSDWAIT, and a branch is taken using register RC.

Location PURGE1 loads parameter register 1 with a value of 56 and branches to location PURGE. Location PURGE loads register

RC with a V-type address constant for TMSPURGE and branches to that location. Location ABEND issues an O/S ABEND macro instruction specifying a user completion code of 777 with a dump to be taken.

Location TMSWRDEQ is another entry point to the TMSCSIO routine entered from the TMSWAIT routine when the TMSWAIT routine finds that more than one terminal has issued a write request at the same time. The entry is directly from the routine TMSWAIT and serves a function of dequeuing the already enqueued write operation for the second terminal. This entry point provides temporary addressability by using the address in register RC. It also provides permanent addressability by loading the base register, Register RB, with the value of an address constant specifying the location TMSCSIO. The program flags are reset for a normal exit; addressability to the data event control block, the data control block, and pointers to the message address and the message length are loaded. The waiting-to-write count is decremented by 1, and if it is not 0, a branch is taken to location WRITE to issue the write for this terminal. If the waiting to write count (DECWWCNT) was zero, the data event control block flags are reset to indicate no more functions waiting to write, and a branch is then taken to location WRITE.

TMSRDRST is another entry point directly from the routine TMSWAIT. It is similar in function to the entry point TMSWRDEQ because it is used when the routine TMSWAIT finds that when one function block had issued the read operation, the write was already in progress, and the parameters must have been saved. When the write operation has finished, the routine TMSWAIT uses this entry point to start up the read that was queued at a previous time. It provides temporary program addressability by using the address in register RC, loads the base register RB with an address constant specifying the address TMSCSIO, loads the pointer to both the data event control block and the data control block into registers RDECB and RDCB, respectively, and branches to location READO to set up a new read.

Location TMSCSIOR is another entry point used by TMSWAIT when the end-of-transmission routines indicate an I/O error by setting a flag in the function block (FBXMTERR). This routine also establishes temporary and permanent addressability and loads the pointers to both the data event control block and the data control block into registers RDECB and RDCB, respectively. It then clears the function block event control block to prevent dispatch the next time TMSWAIT is entered, and then tests to find which operation resulted in a permanent I/O error. If it was a write operation, a branch is taken directly to location WRITE11 to reissue the write. At this point the data event control block still contains the information it had when the write was issued. If the operation was a read rather than a write, the sense byte at location DECSENSO is tested for indications that the error was

timeout, lost data, or intervention required. If it was any of these, special processing is needed, so a branch is taken to location CSIOERR1. If it was not one of these, a read retry operation code is moved into location DECTYPE+1. A buffer address is loaded into register RBUF from location DECAREA, and a branch is taken to location READ21 to reinitialize the read operation for a read retry.

At location CSIOERR1 the type field of the data event control block is tested for a bit indicating multiple terminals on this line. If there are not multiple terminals, a branch is taken to location CSIOERR2 to skip the following code. If there are multiple terminals, the active polling count is loaded into register RWORK1, decremented by 1, and restored. The address in pointer is updated to correspond to the terminal list entry that was last being polled, and a write positive acknowledge operation is issued to the terminal. The write positive acknowledge is tested for successful completion, and if it was not successful, the operation is retried 10 times. If it was successful, a branch is taken to location CSIOER12.

At location CSIOER12 an O/S WAIT macro instruction is issued to wait for the completion of this write positive acknowledge. Rather than exiting to the routine TMSWAIT, a wait is issued here because the monitor system must remain in control at this time. Control then falls through to location CSIOERR2 where the read-in-progress and write-in-progress are turned off. A cumulative length counter is cleared and the buffer address obtained from location DECAREA. The buffer is released, and message pointers are set up in register RLNTH and RPTR. The read and write operation flags at location FBRWOP are set to indicate a write operation with pre-erase, carriage return before write, and carriage return after write (FBRWRITE, FBRWPE, FBRWCRBW, and FBRWCRAW). A branch is then taken to location WRITE2 to issue a system message to the screen in question, indicating a permanent uncorrectable I/O error and requesting the reissuance of the last message.

4.6 TMSGMFM Module

The function of TMSGMFM is to obtain or free main storage upon request of a user program. The routine will either obtain a block of storage and append it to the storage chain in the Function Block (FBBLKCHN), or it will free a specified block of storage previously obtained by a TMSGETM and update the storage chain appropriately.

Upon entry, TMSGMFM saves the calling program's registers in a temporary save area. Then immediately upon location of the Function Block, the registers are moved to the FBSAVE area. Next, the GMFM entry flag is set in the FB, and register 1 (RP1) is tested for a zero condition. If register 1 contains zero, the intended operation is a GETMAIN, and the requested size is in register Ø. If register 1 is non-zero, the intended operation is a FREEMAIN, and register 1 contains the address of the area to be freed.

The TMSGETM routine first checks to see if return with a condition code is requested. If yes, a special return flag is set on. The routine then bumps the requested GETMAIN size by 8 bytes to cover the storage chain and issues a standard conditional GETMAIN. If the GETMAIN was successful, the size of the block is stored in the second word of the area, and the previous last entry on the storage chain (FBBLKCHN) is stored in the first word of the area. The address of the newly obtained storage block is then loaded into FBBLKCHN. The address of the first byte available to the user (i.e., the first byte following the 8 byte storage chain) is loaded into register 1, and control is returned to the user.

In returning to the user who obtained the requested storage, the GMFM entry flag is set off, the registers are restored from the FBSAVE area, and control is returned. If core was not available, however, two options are possible. If return is requested, register 15 is loaded with the return code of 4, and control is returned as normal. If return is not requested, control is passed to TMSPURGE with a return code of 8, which will purge the user and indicate insufficient main storage left to satisfy a TMSGETM request.

Upon entry, TMSFREEM decrements the address supplied by the user by 8 bytes. The storage chain is then searched for the resulting address. When the block is found, it is freed, and the storage chain is compressed, deleting the freed block. If the address is not found in the storage chain, control is passed to TMSPURGE with a return code of 12, which will purge the user and indicate that the TMSFREEM request does not specify a legitimate address. Otherwise, return is returned as in TMSGETM.

4.7 TMSGTSLE Module

TMSGTSLE is a service module of TMS designed to simulate O/S QISAM while saving approximately 7K bytes of storage. Like other O/S access method modules, it is loaded by the OPEN routines (in this case, the TMSOPEN routine) and, therefore, is passed dynamically in and out of storage depending on usage by a user program. Like other O/S access method modules, it is coded in a re-enterable fashion. The program is invoked by the use of TMSGET, TMSSETL, or TMSESETL macro instruction.

At entry to the program, register 1 will contain the DCB address and a flag in the top byte to indicate GET, SETL, or ESETL. Register 0 will contain data which vary depending on the option selected.

The TMSGTSLE module has one entry point named TMSGTSLE. The module begins with a store multiple of register 14 through 12 in the user's save area, and then tests register 1 for a minus value indicating SETL function. If register 1 is minus, a branch is taken to location SETLROUT.

At SETLROUT the key address and key length specified for the TMSSETL macro are transferred to register 2, and the DCB address is transferred to register 3. A TMSGETM macro instruction is then issued to obtain an area of core equal in size to the blocksize of the file plus four hundred. These four hundred bytes will contain a save area and work space for other routines. When the save area is created, it is linked via standard linkages to the user's save area, the address placed into register 13, the FB address placed into the first word of the save area, and register 0 and 1 restored from 2 and 3, respectively. The registers 0 and 1 are then stored at location REGØ and REG1 in the work area, the READ macro instructions parameters moved to the work area, and a flag set to indicate no reads as yet. A condition code of hexadecimal 80 is stored at location ABSRDIND to be used later in an execute of a branch on condition instruction. This is equivalent to a condition of equal. Location REGØ + 1 is then tested for zero. A zero here indicates that option B of TMSSETL was requested, and location ABSRDIND is changed to indicate unconditional branch (hex 'F0'). Next, the logical record length and blocksize are loaded into registers 2 and 5, respectively, and the blocking factor is computed by a divide. The result is stored at location BLKFACT. Then the key length, blocksize, and buffer address are loaded into registers 5, 3, and 6, respectively. If option B was indicated, the key area is zeroed, the address is loaded into register 9, and a branch is taken to location 21. If option B is not selected, the address of the key (stored at REGØ) is loaded into register 11 and the key length (decremented by 1) is used to move the key to the key area at location KEYBUMP. Control then falls through to location L1.

At location L1, the DCB address is located from REG1, and the address of the work area (called LOCAL) is stored in a TMS-supplied fullword at location 240 of the beginning of the DCB. At location RETURN, the parameter registers are stored in the work area save area, register 13 is restored, and control then falls through location OUT to a TMSRETURN macro instruction.

If, at entry to TMSGETSLE, register 1 is not minus, the address of the work area is obtained by loading register 10 from the address contained in register 1 plus 240. Local addressability of the work area (LOCAL) is established by using this address. The top byte of register 1 is then tested for the presence of a hexadecimal 40. If it is not present, a branch is taken to the ESETL routine at location ESETLROU. If present, GET is indicated and a branch is taken to location READOUT.

At location ESETLROU, the save area chain is changed to indicate the last save area, the work area is freed by issuing a TMSFRERM macro instruction, and a branch is taken to location OUT for a return to the user.

At location READOUT the parameter registers are loaded from the work area save area where they were stored at the end of the SETL routine. The read indicator at location RDIND is tested for a hex '01' to indicate if a read has been issued or if one needs to be. If location RDIND equals hex '01', a branch is taken to the routine to locate the next record at location RECFND. If not, a READ macro instruction is issued and tested for completion with a WAIT macro instruction. A normal completion is tested by masking appropriate bits in the DECM. If completion is normal, control is passed to RECFND. If not, control passes to IDERRORS which loads a completion code, and the address of a message and returns to the user.

At location RECFND the key length, or partial key length, is obtained from the work area and used to compare the key to the first key in the block read-in. The code at location ABSRDIND is then used as a branch code. If the B option is specified in the TMSSETL macro, ABSRDIND always indicates a branch to location MATCH. If option K or KC is selected, ABSRDIND indicates branch only on equal condition. If the key is not the same, register 4, containing the number of records per block, is decremented by one and tested for zero. A zero here indicates record not found and an error condition code and message are returned to the user. If register 4 is not zero, the record length is added to the current record pointer to point to the next record, and a branch is taken to location EXE to re-enter the loop to locate the proper record.

If control is passed to location MATCH, then the proper record has been found. Thus, the key no longer matters and so is set to hexadecimal F's. Likewise, ABSRDIND is set to 'FØ' to indicate unconditional branch. Control will now pass directly to MATCH upon

entry if GET is specified. The record being pointed to is then tested for MATCH against a key of hex F's to see if this is the end-of-file. If it is, the address of the users EODAD routine is loaded and a return made to that point. If it does not indicate end-of-file, register 4 is tested to see if it is the last record in the buffer. If not, register 1 is loaded from register 6 to give the record address to the user, register 6 incremented to point to the next record for the next issuance of the TMSGET macro instruction, and the count of records remaining in register 4 decremented by one. At CLOUT register 1 is spaced by 16 to account for the block header, registers Ø and 1 stored at the appropriate place in the save area, and control passed to RETURN to return to the user.

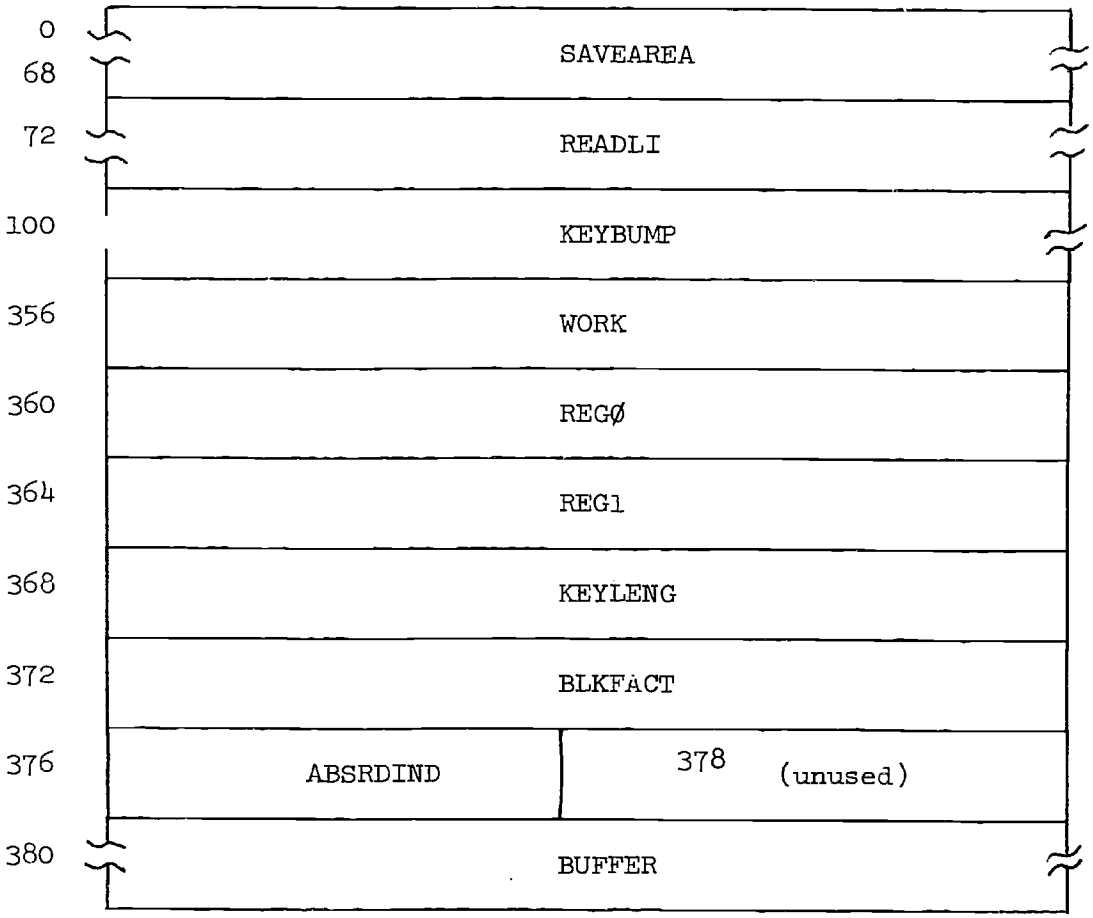
If the count of remaining records indicates the next record in the buffer is the last, control is passed to location LASTREC. Here, location RDIND is set to zeros to indicate that a read will be needed next time, and the key is obtained for the READ by picking up the last key in the present block and adding one to it. The number of records remaining is reset to the blocking factor, the pointer to the current record loaded, and the address in register 6 set to the beginning of the buffer. A branch is then taken to location CLOUT to store the registers and return to the user.

4.8 TMSHSKP Module

The TMSHSKP module begins with a standard O/S register save sequence. A save area within the housekeep routine is linked to the save area provided by the O/S monitor. The routine then issues the message:
TMSOOOI ILR TERMINAL MONITOR SYSTEM INITIALIZING
to the computer operator.

The next major operation is the loading of a predetermined subset of access method modules into main storage where they will reside for the duration of TMS operation. The list of modules to be loaded is given as a BLDL list labeled LOADMODS. The first half of the list is a binary count of the number of entries in the list. This count must be altered if the number of entries in the list is changed. The standard list of I/O modules is given as Appendix 4. The load process is initiated by issuing a BLDL specifying the load list as parameter. A completion code of 4 from the BLDL macro indicates that one or more modules are missing. The routine beginning at label MISSING loops through the BLDL list, testing for a zero record field which indicates a missing module. For each missing module encountered, the message:
TMSOOII TMS INITIALIZING ERROR. UNABLE TO LOCATE MODULE xxxxxxxx
is issued to the computer operator (xxxxxxxx represents the name of the missing module). When this loop is complete, the program is abnormally terminated with a user code of 001. A return code of 8 from

FIG. 4
STRUCTURE OF TMSGTSLE WORKAREA (1 PER QISAM DCB)



NAME	BYTES	USE
SAVEAREA	72	SAVEAREA FOR TMSGTSLE
READL1	28	AREA FOR LIST FROM PARAMETERS FOR READ
KEYBUMP	256	AREA FOR KEY
WORK	4	WORK AREA FOR MASKING REGISTER CONTENTS
REGØ	4	ADDRESS OF KEY + LENGTH FOR TYPES K, KC, SETL
REG1	4	ADDRESS OF DCB FOR THIS WORKAREA
KEYLENG	4	NOT USED AT PRESENT
BLKFACT	4	BLOCKING FACTOR FOR THIS FILE
ABSRDIND		INDICATOR FOR BRANCH INSTRUCTIONS (SEE DOCUMENTATION)
BUFFER	(VARIABLE, EQUALS BLOCKSIZE)	INPUT AREA FOR ONE BLOCK FROM FILE

the BLDL macro indicates an I/O error occurred while attempting to read the module directory. Control is passed to the code labeled IOERROR which writes the message:
TMS002I TMS INITIALIZING ERROR. I/O ERROR WHILE READING MODULE DIRECTORY
to the computer operator. The program is then abnormally terminated with a user code of 002. A completion code of 0 from the BLDL macro indicates the successful location of all routines. Control passes to the code at label LOAD which first loops through the BLDL list and then issues a LOAD DE call for every entry in the list.

The next major operation is to establish the communication region (CR). For the purposes of this code, the length of the communication region to be obtained is represented by the value CRLENGTH, defined in the CR dsect. An R-type GETMAIN macro is issued to obtain the core. The core is cleared with a single XC instruction (which implies a CR length of 256 bytes or less). The address of this newly obtained area is placed in RCR, and addressability is established using the CR dsect. Finally, the end-of-queue flag is turned on in CRQFLAGS.

In addition to data sets for each communication line TMS also responsible for providing three other data sets: the system library, the snap, and the system log data sets. The housekeep routine contains the skeleton DCB's for these data sets which are labeled LIBDCB, SNPDCB, and LOGDCB, respectively. The combined length of the three DCB skeletons is represented by the value of DCBLGTHS, which is used to provide the length specification for the R-type GETMAIN macro instruction. If the core is obtained, a loop is employed to move the skeletons into upper core a doubleword at a time. Space is provided at label OPENLIST for pointers to the three DCB's mentioned above plus up to 100 pointers to line DCB's. The pointers to the library, snap, and log DCB's are updated within this open list and also placed in their proper positions in the CR. Register RPTR is initialized to point to the current last entry of the open list and will be updated as line DCB addresses are added to it.

The most important operation of the housekeep routine is to set up terminal control blocks and buffers in main storage for use by the TMS execution-time routines. Both the skeletons for these terminal control blocks and certain parameters are supplied by the TMSBLOCK module. The combined length of all terminal control block skeletons is obtained via the entry point TMSBLGTH. As before, this length is used in an R-type GETMAIN macro instruction. Once the core is obtained, the start of the combined terminal block skeletons is located via entry point TMSBLOCK, and the terminal block skeletons are moved into upper main storage a doubleword at a time. The rest of the processing consists of proceeding through the various chains defined in the terminal control

block skeletons and updating all addresses to point to the new copy of terminal control block skeletons in upper main storage. This means that to every address must be added an increment representing the difference between the copy of the skeletons in upper core and the original location of the skeletons. This increment is computed and placed in the register RINCR. The incrementing process begins by finding the original address of the last FB in the chain via the entry point TMSLSTFB in module TMSBLOCK. This address is incremented and stored in the CRFBCHN word in the communication region. Since this now gives the address of a new FB, the various pointers within the FB are incremented. As each new FB is encountered, a halfword in the CR, CRWLX is incremented to accumulate a count of the number of FB's in the system. This will be used later in setting up the wait list. The processing of all FB's is assured by following the chain of FB's down to its conclusion. Whenever we process the first FB for a particular communication line, control is passed to the code, starting at label TBLOCK5, which processes the newly located DECB. This code also increments register RLCT by four to increment the count of communication lines in the system. From here control passes to code for processing the newly located DCB and appending the DCB address to the vector of DCB addresses in OPENLIST. Each address is inserted by turning off the end-of-list bit in the preceding fullword, inserting the address in the current fullword, and turning on the end-of-list bit in the same fullword. The other major operation is to move the DCBBUFCB pointer contents into the associated communicating line DECB and reset DCBBUFCB to x'000001'. This is done to prevent the BTAM dynamic buffering module IGG019MS, which is never needed, from being loaded by the OPEN routine. Following DCB processing, control falls through to process those buffers associated with the DCB by updating the buffer chaining addresses. Control is then returned to the middle of FB processing at location TBLOCK2. Processing of FB's continues as described above until the list FB is processed. Control is then passed to location WLIST.

The next major job to be done is to prepare the initial configuration of the wait list and to load the "phantom job". Because they are highly interdependent, these tasks are performed together. First, the length of the list must be determined. The accumulated count of FB's is obtained from CRWLX. This count is incremented by 2: one for the console wait list entry and one for the queuing/wait list entry. The resulting count is multiplied by four to give a result in bytes, which is then incremented by the contents of register RLCT, representing the number of bytes needed for pointers to the communication line event control blocks. The final total is stored in CRWLX to serve as the wait list offset. This total is then multiplied by two to obtain the overall length of the wait list and used in an R-type GETMAIN macro instruction. A loop is performed to zero out the core so obtained a doubleword at a time. A BLDL macro instruction is issued to locate the

directory entry of the "phantom job" module TMSPJOB. In case of error during the BLDL, branches are taken back to the code that loaded the I/O access method modules in order to report the error to the operator and abnormally terminate the program. If the BLDL is successful, a LOAD DE macro instruction is issued to bring the "phantom job" into core. A loop is then performed to move the line event control block addresses into their section of the wait list and to set a flag in the corresponding wait list extension locations in order to show that they represent line event control blocks, not terminal event control blocks. This flag pattern consists of all one bits in the first byte and all zero bits in the remaining three bytes of the fullword.

The last operation is to proceed through the entire chain of function blocks setting up various locations in the FBSAVE area and entering a pointer to each FBECB in the wait list with the corresponding FB address in the wait list extension. The completion bit also is set on in the FBECB to ready the DB for dispatching when execution of the system begins. At the end of this loop, CRWLLAST will be pointing to the last entry in the wait list, and the end-of-list bit will also be set on for the last entry in the wait list. The effect of all this is to prepare all function blocks to begin execution with registers 12 (RB), 14 (RR), and 15 (RC) pointing to the start of the "phantom job", register 10 (RCR) pointing to the communication region and register 11 (RFB) pointing to the function block. This is the initial execution state for every terminal in the system.

The next operation to be performed is the OPENing of all DCB's currently defined in the system, coupled with a certain amount of post-OPEN processing. The open parameter list labeled OPENLIST has been completely prepared by previous code. All that is necessary to open DCB's in parallel is to issue the OPEN macro instruction, with this list as the parameter. Post-OPEN processing consists of a loop to make one more pass through the chain of all FB's. For every FB that is the first on the chain on its communication line, code is entered to both move the buffer pointer from the DECB to the DCBBUFCB pointer and clear the DECB for use by the system. This reverses the earlier action which prevented the BTAM dynamic buffering module from being loaded inadvertently at OPEN time. The remainder of the process is executed only if the FB represents a terminal on a multi-drop line. In this case the BTAM operation "send ack" is initiated to send an EOT character down the line to initialize the remote devices (in the present case, the Sanders displays). If for any reason the attempt to issue the BTAM write operation is not successful, the program abnormally terminates with the user code of 003, which indicates further difficulty in the line. The program waits for a positive response from this operation before proceeding on to the next FB.

The final operations are to inform the operator that the

terminal monitor system is beginning its normal operation and to transfer control to the executor portion of the system. The message:
TMS003I TLR TERMINAL MONITOR SYSTEM STARTED
is written on the console typewriter. The communication region address is placed in register 1 (RP1). The pointer to the TMS library DCB is placed in the XCTL parameter list, and the XCTL macro instruction is invoked. Thus exit from TMSHSK causes the TMSEXEC load module to overlay directly the housekeep routine and begin execution.

4.9 TMSOPEN Module

TMSOPEN is a module of TMS designed to construct and open a user data control block with only a knowledge of the data set name. Therefore, TMSOPEN differs from O/S standards which require a DD name, the access method, and the MACRF parameter of the DCB.

The module begins with a standard store multiple into the user's supply SAVE area. The address of the user's function block is obtained from the first word of his save area, which provides addressability to his FB. The store registers are then copied from his save area into location FBSAVE, and the user's save area is thereby freed for use as the save area of the TMSOPEN routine. The entry point to the module is obtained from register RC, loaded into register RB, and used to provide addressability for the programmer. The address of the communication region is loaded into register RCR from location FBCE and used to provide addressability for the communication. Register RP1 is loaded into register RPARM and used to provide addressability for the remote parameter list. At this point, the task is to search for a catalog entry for the data set names specified by the user. The location DSNAME is cleared to blanks, and the address of the data set name is obtained from the parameter area. Likewise, the length of the data set name is obtained from the parameter area. These two values are loaded into registers RPTR and register RCTR, respectively. If the specified access method is EXCP with appendages required, a special branch is taken to location CSEARCHA. At this point, the data set name and the data set length addresses are loaded from special addresses in the parameter area into the proper registers.

Control then falls through to location CSEARCHB, where the length of the data set name is tested for validity. If the length is zero or negative, a branch is taken to location DISASTER. Otherwise, the length is used in an execute instruction to move the data set name to the parameter area for the catalog search. The parameters specified by the users are tested to see if the data set is qualified by a user name. If not, a branch is taken to location CSEARCH1. If it is to be qualified, a period is moved into the first location after the data set name, and the name currently in the user's function block is appended to the data set name. Control then passes to location CSEARCH2. At location

CSEARCH1 the parameter flags are tested to see if the user wants to qualify the data set name by the function block number. If not, control is passed to location CSEARCH2. If it is to be qualified, the characters'.FB' are moved to the first positions after the data set name. Then, the terminal number from location FBTERMNO is moved to the first position following the characters '.FB'. Control then falls through to location CSEARCH2.

At location CSEARCH2, a catalog search is done by issuing an O/S LOCATE macro instruction specifying a parameter list called BYNAME. The results of this macro instruction are tested upon return by a branch table. If the return code is zero, then a catalog entry has been found so a branch is taken to location CSEARCH3. Branches are taken to locations DISASTER or CMISSING if the correct volume was not found, the entry was not found, the final entry was not the data set name, or various other error conditions exist. At location CSEARCH3, a search is made to see that all volumes for this data set are mounted. The location TIOTPTR is tested to see if a pointer to the task I/O table already exists. If not, this address is found by the use of an O/S EXTRACT macro instruction. Control then falls through to location CSEARCH4 where the volume count is loaded from the catalog entry at location WORKAREA. A pointer is loaded to point to the first volume entry, and another pointer is set to point to the first DD entry. These pointers are EPTR and RTIOT, respectively. Since, under the Terminal Monitor System, DD names are names to correspond to the volume they specify, DD name ILR04 will specify a volume ILR04. A match merely has to be done between the volume name specified in the catalog and the DD name currently pointed to by register RTIOT. Once the volume serial number matches the DD name, control is passed to location CSEARCH7. If the volume serial number does not match the DD name, the register RTIOT is bumped to point to the next DD entry. A fullword of binary zeros at the address currently pointed to by register RTIOT indicates that an entry does not exist. Therefore, there is no DD card given for the volume requested. A branch is then taken to location CMISSING to indicate that this volume is gone. If a DD entry exists, control is passed to location CSEARCH6 to try again to match the volume serial number to the DD name.

At location CSEARCH7, after a check to see that the volume is mounted properly, a test is made, through the use of an O/S OBTAIN macro instruction, for the existence of a data set control block with the name specified by the user. A branch is taken using the return code, as an index to a branch table, to either location CSEARCH if the data set control block is found, or variously, to locations DISASTER or CMISSING, depending upon the error conditions. At location CSEARCH the pointer to the volume entry in the catalog entry is bumped to point to the next volume entry. Register RCTR, containing the number of volume entries, is decremented by one, and a branch is taken to location CSEARCH5 to establish that this

volume is mounted. Once the volume count in register RCTR becomes zero, control passes through to a section of code which will compute the required lengths of the data control block based upon the access method requested. The first test is to see if the access method requested was EXCP. This is done by testing location PMACRF in the parameter area supplied by the user for the presence of a hexadecimal 80 (PBITE). If this bit is not present, a branch is taken to location FORMDCB1 to test for the next access method. If it is present, register RWORK1 is loaded with a value of 56 for the 52 bytes for the EXCP access method data control block plus 4 bytes for the data control block chainword required by the Terminal Monitor System. Register RWORK2 is cleared to zero, and location PMACRF is tested to see if appendages have been requested for this access method. If not, a branch is taken to location FORMDCB5 to obtain the core for the DCB. If appendages have been requested, a length of 20 is added to the value in register RWORK1, and a branch is taken to location FORMDCB5.

At location FORMDCB1, location PDSORG is the remote parameter area is tested for the presence of a hexadecimal 80 (PBITIS) to indicate the request for the index sequential access method. If this bit is not present, a branch is taken to location FORMDCB2 to test for the next access method. If the index sequential access method has been requested, register RWORK1 is loaded with a value of 244 for the 236 bytes required for the index sequential access method, plus 4 bytes for the chainword, plus 4 bytes used by the Terminal Monitor System module TMSGTSLE. Register RWORK2 is loaded with a value of 16 for the negative offset of the DCB. A branch is then taken to location FORMDCB5 to obtain the core for this DCB.

At location FORMDCB2, PDSORG is tested for the presence of a hexadecimal 20 (PBITDA) indicating the direct access method. If this bit is not present, a branch is taken to location FORMDCB3. If the direct access method has been requested, register WORK1 is loaded with a value of 92, register RWORK2 is loaded with a value of 16, and a branch is taken to location FORMDCB5. At location FORMDCB3, PDSORG is tested for the presence of a hexadecimal 40 (PBITS) indicating that the physical sequential access method has been requested. If this bit is not present, a branch is taken to location DISASTER, since all access methods have now been tested. If this bit is present, register RWORK1 is loaded with a value of 92, and register RWORK2 is cleared to zero. Control then falls through to location FORMDCB5.

At location FORMDCB5 register RPC is loaded from register RWORK1, and then an O/S GETMAIN macro instruction is issued specifying that the length of core to be obtained is now in register 0. The address of the core obtained is to be placed into location COREADDR. Upon the return from this macro instruction, the

return code in register RC is tested for a successful completion, and if register RC is not zero, a branch is taken to location NOCORE4 to indicate core not available. Control passing to the next instruction implies successful completion, and register RDCB is loaded with the address at location COREADDR. Register RWORK1 is stored at location DCBELMEN is case the length of this data control block could be used in an emergency flush. Register RWORK1 is incremented by 1 and used in an execute instruction to clear the obtained core to zero. From location FBDCBCHN the pointer to the next data control block is moved into the chain element word pointed to by register RDCB. The negative offset in register RWORK2 is placed at the top byte of the word pointed to by register RDCB. Register RDCB is then placed at location FBDCBCHN, and the chaining of this DCB with the other DCBs connected to this function block is complete. Once the data control block offset has been subtracted from it, register RDCB is used to provide addressability to the data control block through an O/S DCBD macro instruction.

At this point it is necessary to obtain the Job File Control Block (JFCB) for the volume or volumes upon which the data set resides, and O/S GETMAIN macro instruction is issued specifying a length value of 464. This is to test whether enough core remains for the RDJFCB. Upon return from the GETMAIN, register RC is tested for zero, which indicates normal completion of the GETMAIN macro instruction. If register RC is not zero, the GETMAIN is not successful, and a branch is taken to location NOCORE3. If the GETMAIN is successful, a FREEMAIN macro instruction is issued to free the core obtained by the GETMAIN. The Data Set Control Block for the first volume of the data set then is reobtained using an O/S OBTAIN macro instruction. If this macro instruction is not successful, a branch is taken to location DISASTER. If it is successful, a dummy DD name is constructed at location DSVOLS first by clearing this location to blanks and then by moving in the first volume number from location VOLUME. Location WORKAREA is tested to see if there is more than one volume. If there is only one volume, a branch is taken to location FNDJFCB1 to find the job file control block. If there is more than one volume, location DSVOLS plus 4 is shifted over one byte so that the volume number of the next volume upon which this data set resides may be appended to this volume number (i.e., if the data set is resident on the volumes ILR03 and ILR05, the DD name constructed would be ILR35). This is accomplished by obtaining the number of volumes and looping through to append the next volume number until the count of volumes in register RC is diminished to zero. Control then falls through to location FNDJFCB1.

At location FNDJFCB1 the DD name at location DSVOLS is moved to location DCBDDNAM in the data control block. The address of the exit list and the open flags are placed into the DCB, and the

DCB address is placed into location OPENPARM. The top byte of location OPENPARM is set to a hexadecimal 80, indicating the end of the open list. An RDJFCB macro instruction has been issued to bring the Job File Control Block for this specified DD name into core as an addressable location. Addressability to the Data Set Control Block (DSCB) and the Job File Control Block (JFCB) is then provided through the use of USING assembler instructions. Next, the data set name is moved into the JFCB, and if the data set is qualified, a branch is taken into location SETJFCB1. If it is not qualified, location JFCBIND2 is set to indicate a shared data set (JFCBISHR), and a branch is taken to location SETJFCB2. At location SETJFCB1, JFCBIND2 is set to indicate an old data set (JFCBIOLD); control then falls through to location SETJFCB2.

At location SETJFCB2, the data set organization indicator is moved from location PDSORG in the user's supply parameter area to location JFCBSORG in the Job File Control Block. The volume count of this data set is loaded into register RCTR from location WORKAREA in the DSCB for this data set. This number is stored both in location JFCBNVOL, which holds the number of volume serial numbers, and in location JFCBVLCT. Using the count in register RCTR, the volume serial numbers are then moved from location WORKAREA + 2 in the data set control block work area to location JFCBVOLS by looping through decrementing register RCTR until register RCTR is zero. The key length for the data set is then moved from location DS1KEYL in the DSCB to location DCBKEYLE in the data control block. The data set organization is likewise moved into the data control block at location DCBDSORG, and the end of the data address is moved from the parameter area from location PEODAD to the data control block at location DCBEODAD. The record format for this data set is moved from location DS1RECFM in the Data Set Control Block to location DCBRECFM in the data control block. The data set organization specified in the user's parameter area at location PDSORG is tested for an index sequential data set (PBITIS). If this is not an index sequential data set, a branch is taken to location SETDCBO. If it is, location PMACRF + 1 in the user's supply parameter area is tested to see if the TMSGTSLE flag is set (a hexadecimal 80). If it is not set, a branch is taken to location SETDCBO. The setting of this flag means that the user is going to employ the queued index sequential access method, which requires the loading of the service module TMSGTSLE. If the flag is set, location PMACRF (the user's macro instruction references) is moved to location DCBMACR in a data control block. Location DCBMACR + 1 is ANDED with a hexadecimal value of 7F to turn off the TMSGTSLE flag in the data control block. This flag is used only by the TMSOPEN routine. A standard linkage is then set up to the routine TMSPLOAD with register 1 pointing to a location containing the name TMSGTSLE. Before a linkage to this routine is established, the registers saved by TMSOPEN at location FBSAVE are temporarily stored in location TEMPSAVE. Then a standard BALR instruction is issued. Upon return from the TMSPLOAD routine, the registers at location

TEMPSAVE are restored to location FBSAVE, and the success or failure of the loading of the routine TMSGTSLE is determined by testing register RC for the presence of a zero. If register RC is not zero, the program was not loaded successfully, and a branch is taken to location NOCORE3. If it is successfully loaded, the entry point of the routine currently in register RPO is stored at location CRGTSLE in the communication region. A branch is taken to location SETDCBØ + 6 to avoid repeating the macro reference field's move to the data control block.

At location SETDCBO the macro instruction reference parameter, which the user supplied at location PMACRF in the parameter area, is moved to location DCBMACR in the data control block. Location PMACRF is tested for the presence of a bit indicating EXCP access method (PBITE). If it is present, a branch is taken to location SETDCB2. Otherwise, the option codes at location POPTCD are moved to location DCBOPTCD from the user's parameter area to the data control block. Likewise, the synad routine address supplied by the user at location PSYNAD is moved to location DCBSYNAD. If this is not an index sequential data set, a branch is taken to location SETDCB1. If i. is an index sequential data set, the numeric portion of location PARMFLGS is moved to location DCBMAC for the index sequential macro instruction reference extension. Then, the relative key position is moved from location DS1RELKP in the Data Set Control Block to location DCBRKPN in the Data Control Block. At location SETDCB1 the blocksize is moved from location DS1BLKL to location DCBBLKSI (all locations starting with the prefix DS1 indicate that this location is in the Data Set Control Block; likewise, locations starting with the prefix DCB indicate that this location is in the Data Control Block). The data set organization at location PDSORG is tested for a bit indicating a direct access data set (PBITDA). If this is a direct access data set, a branch is taken to location SETDCB3. If not, the logical record length is moved from location DS1LRECL to location DCBLRECL, and then a branch is taken to location SETDCB. At location SETDCB2, location PMACRF is tested for a bit indicating that appendages are required with the EXCP access method (PBITAPP). If appendages are not required, and this bit is not present, a branch is taken to location SETDCB3. Otherwise, location POPTCD specifying the option code is moved to location DCBOPTCD; likewise, the appendage identification codes are moved from location PAPPIDS to location DCBEOEA. Control then falls through to location SETDCB3.

Here location PMACRF is tested again to see if the EXCP access method is specified and, if so, a branch is taken to location OPEN. Otherwise, location PARMFLGS is tested to see if the user wants to prevent buffer generation (PNOBUFFS). If the user does want buffer generation prevented, a branch is taken to location BUFFER4. Otherwise, the blocksize for this data set is loaded into register RWORK1 from location DCBBLKSI. The quantity now in

register RWORK1 is made into an integral number of doublewords by appropriate right and left shifts. Location PDSORG is tested for a bit indicating an index sequential data set. If this is not an index sequential data set, a branch is taken to location BUFFER1. If it is an index sequential data set, the key length of the record is obtained. Ten bytes for the length field and 15 bytes for the padding are added to the key length and then rounded down to a multiple of 8. This quantity is added to the basic buffer length in register RWORK1, and control falls through to location BUFFER1. At location BUFFER1, the buffer length currently in register RWORK1 is stored at location DCBBUFL. The quantity in register RWORK1 is incremented by 8 bytes for the buffer control block, and buffer alignment is tested by examining location DCBBFLAN to see if the user has specified fullword, rather than doubleword alignment. If the user has not specified fullword alignment, a branch is taken to location BUFFER2. If he did specify fullword alignment, register RWORK1 is again incremented by 8. Control then falls through to location BUFFER2 where the buffer size is transferred to register RPO, and a GETMAIN macro instruction is issued specifying that this length be obtained. The success of this GETMAIN is measured by testing register RC for a return code of zero. If this register does not contain zero, a branch is taken to location NOCORE3. If the GETMAIN was successful, the address of the core obtained is stored at location DCBBUFCB, and both the buffer control block and the pointer field of the first buffer are cleared to zeros. A buffer count of 1 is placed in both the buffer control block and in the data control block at location DCBBUFNO. The buffer length is moved from location DCBBUFL to the buffer control block. The address of the first available byte past the buffer control block is obtained and put into register RWORK1. Once again, buffer alignment is tested, and if fullword alignment is not requested, a branch is taken to location BUFFER3. If it is requested, the address of the first available byte is incremented by four to point to the first fullword past the buffer control block that is not also doubleword alignment. Control then falls through to location BUFFER3 where the address of the first available byte is stored, not only in the first fullword of the buffer control block, but also at location FBSAVE plus 8 corresponding to register RPO. A branch is then taken to location OPEN. If, at location BUFFER4, the user specifies prevention of the buffer generation, a hexadecimal 01 is moved to location DCBBUFCB + 3 to indicate no buffers. Control then falls through to location OPEN where a GETMAIN is issued to see if enough remains for the resident OPENJ processing routines. The normal completion of these GETMAINs is determined by testing for the presence of a 0 return code in register RC. If the return code is not 0, a branch is taken to location NOCORE3. If it was successful, a FREEMAIN macro instruction is issued to release the core obtained by the GETMAIN, and then an OPEN macro instruction type J is issued. On return from

this macro instruction, DCBOFLGS is tested for a hexadecimal 10, which indicates that the OPEN was unsuccessful. If there is no hexadecimal 10, a branch is taken to location DISASTER to indicate the failure of the OPEN. However, if it was successful, register RDCB containing the address of the data control block is stored at location FBSAVE + 12 corresponding to register RP1. Location PMACRF + 1 is tested for a hexadecimal 80, which indicates that this data control block is for the queued index sequential access method. If this bit is not present, a branch is taken to location RETURN. If it is present, the top bit of location DCBMACRF + 1 is turned on to tell the CLOSE routines that the module TMSGTSLE is resident and must be deleted. Control then falls through to location RETURN, where a return code of 0 is loaded into register RC. Control then falls through to location RETURN1, where register RC is stored in location FBSAVE + 4, which corresponds to register RC. A standard return of LM and BR instructions are used.

At location CMISSING corresponding to location VMISSING, register RP1 is loaded with a value of 16, register RC is loaded with a return code of 4, and a branch is taken to location PURGE. At location NOCORE1, the core for the buffer pool must be freed. The necessary freeing of the core for the buffer pool is done at location NOCORE1 by obtaining the pointer to the buffer pool from location DCBBUFCB; finding from this location the number of buffers; multiplying this number of buffers by the buffer size; testing for fullword alignment and, if fullword alignment was specified, adding 8 bytes; adding 8 bytes for the buffer control block; and freeing the core of this size with the use of a FREEMAIN macro instruction. Control then falls through to location NOCORE3, where the DCB length is loaded from location DCBELMLN. The address of the data control block from location FBDCBCHN, the data control block chain, is updated by moving the current entry in the chain to location FBDCBCHN. A FREEMAIN macro instruction is then issued for the core held by the data control block. At location NOCORE4, register RP1 is loaded with a value of 20, register RC is loaded with a return code of 8, and a branch is taken to location PURGE. At location DISASTER, register RP1 is loaded with a value of 24, register RC is loaded with a return code of 12, and a branch is taken to location PURGE.

At location PURGE, location PARMFLGS is tested for a value of hexadecimal 80, indicating that the user wants control to return to himself with the relevant condition code should anything fail in the TMSOPEN routine. If this bit is present, a branch is taken to location RETURN1. Otherwise, register RC is loaded with the entry point of the routine TMSPURGE, and a branch register instruction if taken on register RC.

4.10 TMSPJOB Module

The TMSPJOB module has only one entry point, and its characteristics are somewhat different from those of other entry points. The primary distinction is that permanent addressability is set up in register 12 (RB) prior to entry into this routine. This is done because initial entry into this routine is not made by a specific branch from another routine. Instead, this routine is dispatched for this particular FB as if some third routine has caused the posting of a wait, simulated to have occurred just before the beginning of the phantom job code.

Since the base register, the FB pointer, and the CR pointer are all assumed to have their proper contents when entering TMSPJOB, addressability is established immediately by means of the necessary USING instructions. R-type GETMAIN then is issued for a 72-byte area which will become the topmost save area for the FB pointed to be RFB. This entire new save area is cleared, and the FB pointer then is stored in the first word so that it may be propagated to succeeding save areas by code generated by the TMSSAVE macro instruction. The TMSCSIO macro instruction is then invoked to write the message:
TMS100I - TMS IN OPERATION
to the terminal associated with the current FB. Control then passes to the next block of code.

The next major section of code headed by the label LOGIN asks for, receives, and then processes the user's identification code. A call is first made to the console I/O routine to issue the message:
TMS101A - WAITING FOR LOGIN
to the terminal involved. This issuance is followed immediately by a request to read a response from the terminal. A check is first made to see if the response is the word DISCONNECT, which indicates that the user wishes the terminal logically disconnected from the system. If this match is true, a branch is made to the code at label DISCONCT which issues a WTO to write on the console typewriter the message:
TERMINAL nn DISCONNECTED BY USER
where nn is the terminal number in ECBDIC which has been generated by the TMSHSKP routine and placed in the function block at FBTERMNO. The terminal disconnected flag FBDSCNCT is set on in FBFLAGS. Then the FBECB first byte is set to all zeros, and the macro TMSWAIT is invoked. Since there is no outstanding operation to post the FBECB complete, this effectively puts the terminal in a permanent wait condition. If the response from the user terminal is not the word DISCONNECT it is assumed to be a user identification code. The last non-blank character in the reply is found. Since this is assumed to be the EOT character, it is set to blank in order not to interfere with comparisons of responses which have three characters or less with the table of authorized user identification codes. The valid user identification codes are found in the list labeled USERLIST. The number of entries in this list is placed RCTC, and

the address of the first entry in the list is placed RPTR. Prior to a check of the user list, however, a pass is made down the FB chain to see whether the user ID that has been sent back is identical to the contents of FBNAME in any FB. If an identical match is found, the message:
TM103I- NOT ACCEPTED, NAME ALREADY IN USE
is sent to the user terminal, indicating that acceptance of the user ID as supplied would result in duplicate concurrent user ID names. A branch is then taken back to LOGIN to reissue the invitation to log-in and read a new response from the terminal. If the user identification code does not duplicate the code found in any other FB, a loop is executed to compare the user ID code against a list of valid user ID codes. If this loop is exited without a match, the message:
TMS102I- NOT ACCEPTED
is sent to the user terminal, and a branch is taken to LOGIN to invite another attempt to log-in. If there is a match, the branch is taken to ULCHECK2 where the user name is moved to FBNAME, and the message:
TMS102I- nnnn LOGGED IN
is sent to the user terminal with the user name in place of nnnn. Control then falls through to the next block of coding.

The next section of coding beginning with the label SPECIFY determines which program the user wishes to execute under TMS and brings it into main storage if there is enough space. This routine first issues the message:
TMS104A- SPECIFY PROGRAM
to the user terminal and then reads the response. When a response is received, a check is made first to see whether the initial 6 characters of the response equal the word LOGOUT. If so, a branch is taken to location LOGOUT, where the message:
TMS105I- nnnn LOGGED OUT
is sent to the user terminal with the user name replacing nnnn. The area FBNAME is then restored to blanks, and a branch is made to LOGIN to invite log-in by the next user. If the response is not the word LOGOUT, the first eight characters of the response are assumed to represent the name of a program in the TMS library. As in the log-in routine, the last character of the response message is changed from EOT to blank in order not to interfere with comparisons. The pointer to the program name in register 1 (RP1) is also copied into register 0 (RP0) to save it. The address of the TMSPLOAD module is then found from the communication region, and a subroutine call is made to this routine. Upon return from this routine, the completion code may have one of four values. A completion code of zero indicates a successful load of the program. Branches are made to the code labeled ENTER. ENTER moves the program name in the reply buffer to the double word FBPNAME to be used later in deleting the program, and then it enters the program as a normal subroutine call. A return code of four from the program load routine means there is not enough main storage to complete the load; in this case a branch is taken to

NOCORE where the message:
TMS109I- NOT ENOUGH CORE TO LOAD PROGRAM
is sent to the user terminal, and a branch is taken to SPECIFY to invite the user either to respond with the name of another program or to log-out. A completion code of eight upon return from the program load routine indicates that a specified load module could not be found in the library. A branch is taken to the code labeled NOMODULE where the message:
TMS107I- PROGRAM NOT FOUND
is sent to the user terminal, and a branch is made either to SPECIFY to invite another try to spell the program name properly or to load another program. A completion code of twelve upon return from the program load routine means that a program with the specified name was already found in main storage but was found not to be a re-enterable module. In this case a branch is taken to the code labeled NONRENT, where the message:
TMS108I- PROGRAM NOT RE-ENTERABLE AND ALREADY IN USE. WAIT OR TRY ANOTHER
is sent to the user terminal, and a branch is made to SPECIFY to allow the user to try another program or log-out. If there is a completion code of 16 from the program load routine, there is a severe input/output error in attempting to load the program. A branch is taken to the code labeled IOERROR, which consists of a halfword of binary zeros. This will force an immediate program check and abnormal termination with a dump.

As mentioned previously, in the normal sequence of events the program is located successfully and loaded into storage, and a subroutine call is made to the program from the TMSPJOB module. Normal termination of the user program consists of a standard subroutine return via a register 14 (RR). A normal return via the TMSPURGE module is a return to a point 4 bytes past the point indicated by the contents of register 14. In TMSPJOB, the two full words immediately following the BALR used to enter the applications program consist of a branch instruction to the code labeled NORMAL followed by a branch instruction to the code labeled PURGED. The code labeled NORMAL checks the pointers FBBLKCHN and FBDCBCHN to see that all storage obtained by the application program has been released and all DCBs opened by the application program have been closed. If either or both of these criteria are not met, a branch is made to the code at PURGE which executes a subroutine call to TMSPURGE with a value of 4 in the error index parameter supplied in register 1. Control then drops through to the code with the label PURGED. If the above criteria have been met, control passes on to code which writes the message:
TMS106I- NORMAL EXIT FROM USER PROGRAM
on the user terminal. Control continues on to code labeled DELETE, which issues a DELETE EPLOC macro instruction with the contents of FBPNAME as parameter, thus deleting the application program and and removing it from main storage if necessary. This delete

operation is followed by both the clearing of FBPNAME to blanks and a branch to SPECIFY to ask the user to either specify a new program or log-out.

There are two ways the TMSPJOB detects that some form of error has occurred during operation of the application program. The first is the return to TMSPJOB from TMSPURGE by an offset of 4 bytes from the normal return point. The other is a normal subroutine return from the TMSPURGE routine that was directly invoked by TMSPJOB at location PURGE. In either case control eventually passes to the code labeled PURGED, which issues the message:
TMS1101- ABNORMAL RETURN FROM USER PROGRAM VIA PURGE ROUTINE
and takes a direct branch to the code labeled DELETE to delete the program that had been loaded.

4.11 TMSPLOAD Module

The function of the TMSPLOAD module is to load a user program at the request of another TMS module. TMSPLOAD begins by storing the calling module's registers in the FBSAVE area and setting the TMSPOLOAD entry flag on. The name of the program to be loaded, which is pointed to by register 1 (RP1), is then stored into the BLDL list. TMSPLOAD then checks to see if sufficient core is available for the BLDL routine (406 bytes). If core is not available, control is returned to the calling routine with a completion code of 4 in register le (RC). Otherwise, the BLDL (SVC8) is issued, and the load list is built. If the BLDL returns with a completion code of 8, control is returned to the calling module with a completion code of 16 in register 15. This indicates that a permanent I/O error was detected during the directory search. If the BLDL returns with a completion code of 4, control is returned to the calling module with a completion code of 8 in register 15. This indicates that the requested module could not be found. If the BLDL returns with a completion code of Ø, TMSPLOAD searches the FBCHAIN for an already-loaded copy of the program. If it finds a copy loaded, and the program is flagged not re-enterable, control is returned to the calling module with a completion code of 12. This indicates that the load request was for a not re-enterable module that was already loaded. If TMSPLOAD finds a copy of the program loaded, and the program is flagged re-enterable, the program is loaded via the load SVC (SVC8), which merely bumps the use count by one. The entry point of the loaded module is then stored in the FBSAVE area for the terminal requesting the load, and control is returned to the calling module with a completion code of zero.

If no copy of the requested program is found in the FBCHAIN, TMSPLOAD checks to see if sufficient core is available to load a copy of the requested program plus the forty (40) bytes required

for the associated request block. If core is not available, control is returned to the calling module with a completion code of 4. Otherwise, the program is loaded via the load SVC (SVC 8), the entry point is stored in the FBSAVE area, and control is returned to the calling module with a return code of zero.

4.12 TMSPURGE Module

The function of TMSPURGE is to delete a user program currently running on one of the terminals, free all storage gotten by the user program and close all files opened by it. Since control will not be returned to the user after TMSPURGE, the user's registers are not saved. Upon entry, the TMSPURGE entry flag is set on. Next the pointer to the message indicating the reason for entering PURGE is set up using the offset received in register 1. The save area chain is then traced to the highest save area, and the registers are restored to their condition prior to entering the program being purged.

PURGE then begins closing all attached DCB's by following the DCB chain (FBDCBCHN) located in the FB and entering TMSPCLOS with all of the DCB addresses on the chain. The end of the DCB chain is indicated when FBDCBCHN equals zero. When all DCB's are closed, PURGE begins freeing all attached storage areas obtained by TMSGETM by following the storage chain (FBBLKCHN) located in the FB.

When all attached DCB's are closed and all attached storage is freed, the error message indicating the reason for entry to PURGE is displayed. This is followed both by computing the relative address of the error detected and by displaying the error location message. Finally, control is given to TMSPJOB with an offset of four from the normal return point to PJOB. This results in deleting the user program and displaying the abnormal return from user program via PURGE routine message, followed by the request to specify program.

The messages issued by TMSPURGE are as follows:

TMS150I - PROGRAM ENDED WITH STORAGE OR DATA SET STILL ATTACHED.

TMS151I - INSUFFICIENT MAIN STORAGE LEFT TO SATISFY TMSGETM REQUEST

TMS152I - TMSFREEM REQUEST DOES NOT SPECIFY LEGITIMATE ADDRESS

TMS153I - ATTEMPT TO OPEN AN UNAVAILABLE/UNCATALOGED DATA SET

TMS154I - INSUFFICIENT MAIN STORAGE LEFT TO COMPLETE OPEN OF A DATA SET

TMS155I - DISASTROUS ERROR IN TMSOPEN

TMS156I - TMSCLOSE REQUEST DOES NOT SPECIFY LEGITIMATE ADDRESS

TMS157I - END OF DATA DETECTED WITH NO EODAD SPECIFIED

TMS158I - SYNCHRONOUS ERROR DETECTED WITH NO SYNAD SPECIFIED

TMS159I - INSUFFICIENT CORE LEFT FOR DEBUGGING

TMS160I - ERROR DETECTED IN TMS. USER PURGED WITH SNAP

TMS161I - ERROR DETECTED IN TMS. USER PURGED, SNAP UNSUCCESSFUL

TMS162I - ERROR DETECTED IN PROGRAM. USER PURGED.

TMS200I - ERROR OCCURRED AT RELATIVE LOCATION XXXXXXXX

4.13 TMSTREND Module

The TMSTREND module has two entry points. The main entry point is TMSTREND, which is used for entry into the program from TMSWAIT when an input-output operation for the communication line is complete. The second entry point, TMSCHEND, is a simple BR return through register 15 (RR), and is not called by any other TMS module.

Since control is passed only to and from other TMS modules, the user's registers are not saved on entry. TMSTREND assumes that on entry register 1 (RP1) points to the DECB for the communication line, and after establishing permanent addressability, establishes addressability for the DECB and corresponding DCB for the line, the FB, and the CR.

After initialization of the appropriate base registers, the DECB is checked to see which operation was in progress so that either the address of the polling characters may be found for a read, or the addressing characters may be found for a write. In either case, unless there is only one terminal, a skip bit is set in the current polling entry whose address is found at location DECPOLPT or DECADRPT in the DECB.

The next section of code, starting at location CEENTRY2 after the skip bit has been set, locates the FB for which the operation is complete. This is done first by loading register RWORK2 with the address of the terminal list from location DECTLIST in the DECB, and then subtracting it from register RWORK1, which was previously loaded from either DECPOLPT or DECADRPT. The result in RWORK1 is the offset to the terminal list entry, which is compared to the offset specified at location FBTLOFF in the current FB being processed. If equal, the proper FB has been found, and a branch is taken to CEENTR4. If it is not the proper FB, a loop is executed to follow the chain of FB's to the end, checking each one. If no proper terminal offset is located, an ABEND is executed with a user completion code of 777, and a code of 50 is placed at location CRABCODE in the CR.

Once the FB has been located, the buffer address is loaded into register RBUF from DECAREA and documented by four to point

to the buffer control word. Location DECUFLAGS is then tested to find which operation was in progress, and a branch is taken to either CEREAD or CEWRITE for corresponding read or write operations. If DECUFLAGS indicates either both or neither operation in progress, a 51 or 52, respectively, is placed in CRABCODE, and ABEND is issued.

The code at location CEREAD begins by checking that the flags in the buffer control word indicate buffer-in-use (BUFFINUS) and buffer-waiting-for-input (BUFFWTIN) before proceeding. If these flags are not set, a 53 is placed in CRABCODE, and an ABEND is issued. Then location DECFLAGS is tested for negative response (DECFNEGR) which indicates that channel end is due to polling reset. If this flag is on, a branch is taken to CEREAD5; if this flag is off, there is an incoming message. Register RBUF is then incremented by four to point past the buffer control word to the start of the message. The maximum length of the message is loaded into RCTR from DECLNGTH, and the residual count at DECCOUNT is subtracted from it to find the actual length stored at FBLMLNTH in the FB. This length then is used also to translate the incoming message. If there are multiple terminals, the pointer to the message is spaced over the header and the length adjusted by four.

At CEREAD3 the buffer address is stored in FBBUFPTR, and the text offset is computed by a simple subtract and stored in FBBUFOFF. The buffer flags indicating waiting for input are turned off (BUFFWTIN), and buffer attached to FB (BUFFATFB) are turned on. The DECUFLGS are reset to turn off read in progress (DECUFRIP) and polling reset (DRCUFPRS). The active polling count is reduced by one if there is more than one terminal, and a positive acknowledge is written to the sending terminal. A return is made to entry point TMSDWAIT in the TMSWAIT module. If there are not several terminals on the line, at location CEREADY, the completion code is moved to location FBECB. Location DECSDECB then is set to zero, and a return made to entry point TMSDWAIT.

If channel end is due to polling reset, control passes to location CEREAD5 where flags at DECUFLGS are tested for read-in-progress, waiting-to-write and polling-reset (DECUFRIP, DECUFWTW, and DECUFPRS). The negative response flag, skip bit, read-in-progress flag, and polling-reset flag are turned off, and the polling interrupted (DECUPIN) flag is turned on. The line ECB is zeroed, and the operation type, buffer length, buffer address, terminal polling list address, and relative line number are stored at location DRCRSAVE to be available for later polling restart. The dummy ECB in the CR is then set to indicate operation complete and line available for write, and a return to TMSDWAIT is made.

If the operation tested at CEENTRY4 is a write, control passes on to location CEWRITE. The type of operation flag at location DECTYPE+1 is tested to see if the operation is a write positive

acknowledge. If it is, control passes to CEWRITE7, where the write in progress and acknowledge flags are turned off. The completion code is moved to location FBECB, the line ECB is zeroed, and a branch is taken to CEWRITE3. If the operation is not a write positive acknowledge, the completion code is moved to the FB ECB, and the line ECB (DECSDECB) is zeroed. The buffer pointers are reset and the buffer is filed with blanks. The write in progress flag is reset and, if a transmission error is being processed, control is passed to CEWRITE7 to process as a positive acknowledge. Otherwise, at CEWRITE3 there is a test to see if a write is queued for the line by marking location DECUFLGS with DECUFWTW. If a write is queued, a return is made to TMSDWAIT. If a write is not waiting, DECUFLGS is marked to test for polling interrupt. If polling was interrupted, control moves to CEWRITE5. If not, the active polling count (DECAPCNT) is tested for zero. If there are no other reads in progress, a return is made to TMSDWAIT; otherwise, a return is made to the read polling restart routine (TMSRDRST).

At CEWRITE5 the polling interrupt flag is turned off, and the operation code, buffer length, buffer address, terminal polling list address, and relative line number are restored to the data event control block. The read-in-progress flag is set, and a READ macro instruction is issued for the line. Control then returns to TMSDWAIT.

If, at CEENTRY4, a transmission error is detected, control passes to location CEERROR. If the error already is being processed, and it is the second time through, control passes to an ABEND macro instruction with a 58 placed at location CRABCODE. The data event control block channel status word status is tested for channel end, device end, and unit check flags, and if not present, control is passed to ABEND with a 59 in CRABCODE. The data event control block first sense byte (DECSENS0) is tested for timeout, lost data, or data check and, if any are indicated, a branch is taken to CEERROR2. If none are indicated, control passes to ABEND. At CEERROR1, the transmission error flag at location FBFLAGS (FBXMTERR) is set, and the read or write in progress flag and the skip bit are turned off. The FB ECB is then posted complete with the error, the line ECB is zeroed, and a return is made to TMSDWAIT via CERETURN.

The return and location CERETURN is a simple load of a V-type address constant specifying the entry point TMSDWAIT into register RR, followed by a simple branch register on register RR.

4.14 TMSWAIT Module

The TMSWAIT module has two principal entry points. The first entry point, TMSWAIT, is used for entry into the module from application programs. A subsidiary entry point, TMSDWAIT, is used as a direct entry into the wait module, bypassing certain

register saving and restoring conventions used in calls from application programs. Depending on the status of the wait list, this module may exit to an application program - the TMSCNSL module to process a message from the computer operator, the TMSCSIO module to initiate console input-output for a newly-freed line, or the TMSTREND module to perform end-of-transmission processing for a communication line which has just finished its I/O task.

On entry through entry point TMSWAIT from an application program the contents of registers 14 (RR) through 12 (RB) are stored temporarily in the save areas pointed to by register 13 (RS). The address of the RB is obtained from word zero of this save area. Once FB addressability is established, the contents of the registers which were saved upon entry are moved to the corresponding FBSAVE. At this time the contents of register 13 are also saved in FBSAVE. Permanent addressability to the wait module is established, and the CR address is loaded into register 10 (RCR) from FBCR. The code beginning at location ENQUEUE places the ECB address supplied in register 1 (RP1) onto the end of the wait list. This is done by loading the address of the last wait list entry from location CRWLLAST into register FLAST, incrementing by 4, and using the resulting address to store the new ECB address into the wait list. The contents of CRWLX, the wait list offset, are added to RLAST to find the correspoiding area in the wait list extension. The FB address is stored in this area. The code beginning at location WAIT sets the end-of-wait-list indicator into the high order byte of the current last word of the wait list. The pointer to the wait list in area CRWL is put into register 1, and the WAITR ECBLIST macro is issued to relinquish control of the computer if no operation has been completed.

Either immediately or when one wait condition is satisfied, control falls through the code labeled ENDWAIT. This begins to search the wait list for the first completed event control block. The first operation is to reset the end-of-wait-list indicator. The console ECB address is then loaded into a register, and the setting of the completion bit is tested. If this bit is on, a branch is taken to location CONSOLE. The code at this area both loads the address of the system save area (entry point TMSSYSSB in the TMSBEGIN module) into register 13 (RS) and takes a standard entry into the TMSCNSL module via the entry point of the same name.

If the console ECB is not yet complete, the queuing ECB is tested. If the completion bit is set in this ECB, a branch is taken to location FBQPROC. The code at this point employs repeated calls to subroutine DEQUEUES both to find the first FB on the FBQ chain waiting for newly-freed resource and to remove that FB from the Q chain. This is done as follows: the resource freed for use is identified by a bit in location CRECB. When this bit is found set, three masks are set up for the use of the dequeuing subroutine.

They are QFLAGCR for testing location CRQFLAGS, QFLAGFB for testing location FBQFLAGS, and UFLAGDEC for testing location DECUFLAGS. The subroutine DEQUEUES is linked to, using RRET as an internal return register. Upon return from this subroutine, register RFB is tested for non-zero. A non-zero return indicates a successful dequeuing and a branch is taken to the appropriate routine. A zero indicates that no FB was found to be queued on the free resource. In this case the corresponding flag is turned off in CRECB, and a branch is taken to the next test. Two tests of the FB queue chain are now implemented. The first is for a line now free for a write operation. If the FB requiring this resource is found, a branch is taken to the entry point TMSWRDEQ in the TMSCSIO module to initiate writing on the line. The other test that is now made is for a line that is free for a read operation. If an FB queued on this resource is found, a branch is taken to the TMSRDRST entry point of the TMSCSIO module to initiate a poll restart. In the event that any of the preceding tests do not succeed, there is an error condition, and the first byte of location CRECB is cleared to all zeros. A branch then is taken back to location WAIT to continue processing the wait list.

After the special processing of the first two ECB addresses, the remainder of the wait list is searched for the first ECB with these completion bit sets by means of a simple loop. The completion bit is set on for at least one ECB, since failure for this being done would indicate a gross error on the part of the operating system. Therefore, whenever further wait list processing is to be done, and if it is not certain that there are any further ECBs with completion bit set, return should be made to location WAIT for a further reissuing of the WAIT macro to O/S. When the first completed ECB is found, it is necessary to determine whether this ECB represents a physical communication line or a logical terminal. This is done by checking the first byte of the corresponding fullword in the wait list extension for a bit pattern consisting of all one bits. If this pattern is found, the ECB in question represents a communication line, and entry is made to the TMSTREND entry module via the entry point of the same name to process the end-of-transmission condition. If this special bit pattern is not found, a branch is taken to code at location DEQUEUE, which loads the address of the corresponding FB into RFB from the wait list extension. If necessary, all ECB addresses in the wait list and their corresponding FB addresses in the wait list extension are moved up to fill the space vacated by removel of the ECB and FB addresses from the wait list. Following this, register RLAST is decremented by 4 to reflect the shortening of the wait list. Finally, the contents of all registers are reloaded from the proper FBSAVE, and the return to the application program is taken through register 14 (RR).

The entry point TMSDWAIT is used for entry into the wait routine from other elements of the monitor system. Its principal

purpose is to avoid the saving of registers in FBSAVE, since entry is not from an application program. After setting up a base register pointing to the beginning of the TMSWAIT module in establishing permanent addressability, the code for this entry point loads register RLAST from location CRWLLAST and branches directly to the code at location WAIT. Upon entry to the wait module via this entry point, register 10 always will point to the CR.

5. DETAILED MACRO DESCRIPTIONS

5.1 FORMFB Macro

The FORMFB macro employs three global variables. The arithmetic variables, &FBNO and &TERMNO, are used to maintain a running count of the number of FB's and terminals that have already been defined by previous invocations of FORMFB. Proper usage of these two variables depends upon an uninitialized arithmetic global variable having value 0 when first used. The global character variable &PREVFB is used to contain the name of the last function block generated by the most recent previous invocation of FORMFB. A test is made to see if this name is null, the initial value of a global character variable. If it is, it is set to the character "0", so that when it is employed in an A-type address constant, the proper result will be obtained.

The first operations in the macro-expansion are to identify the type of terminal being employed and to set certain parameters to be used in the remainder of the expansion. The types currently recognized are: 1) M35D: a model 35 teletypewriter attached via a direct link; 2) 2740B: an IBM 2740 basic terminal; and 3) S720W: a Sanders 720 CRT terminal with horizontal screen and the special extra-width option. The local variables that are dependent upon the terminal type are: &DECTTYP, a local arithmetic variable used in setting byte type flags; &DECCPNL, a local arithmetic variable representing the number of characters per second of carriage travel during carriage return; &DECMAXL, a local arithmetic variable representing the maximum number of lines per page for page-oriented devices; &DECCPLN, a local arithmetic variable representing the maximum number of characters per line for the device; and &LISTTYP, a local character variable representing the form of polling list, to be expanded later in the macro.

A standard Data Event Control Block (DECB) is generated by a list-type READ macro instruction, and a TMS-dependent portion is generated by a series of DC instructions. The Data Control Block (DCB) is generated by using the standard O/S DCB macro instruction; the BTAM ine Error Control Block (LERB) is generated by using the standard O/S BRAM LERB macro instruction; the terminal polling list is generated by using the standard O/S BTAM DFTRMLST macro instruction; and the list of polling/addressing characters is supplied by the LIST keyword operand. Following these tables, the buffers for this line are generated. One buffer is generated for every FB. The buffers are linked together, and a pointer to the head of the link is placed in DCBBUFCB. The length of the buffers is determined by the BUFLGTH keyword operand, whose default value is 256. Exit from the FORMFB macro is made with the global variables properly modified for use in a future invocation of FORMFB.

5.2 TABLES Macro

The TABLES macro instruction is a Sanders-supplied macro which causes generation of an EBDCIC to ASCII-8 translate table and/or the generation of an ASCII-8 to EBCDIC translate table. These tables are used for translating messages sent to and received from Sanders displays.

Name	Operation	
	TABLES	EBCASC=name1 [,ASCEBC=name2] ASCEBC=name2 [,EBCASC=name1]

name 1
Is any symbol valid in the Assembler Language. It will be generated as the name of 256-byte EBCDIC to ASCII-8 translate table.

name 2
Is any symbol valid in the Assembler Language. It will be generated as the name of the 256-byte ASCII-8 to EBCDIC translate table.

The macro instruction first checks to see if both parameters are omitted. If they are, the resulting mnote is 'TABLES NOT GENERATED, PARAMETERS MISSING'. If one or both parameters are present, the macro checks for the absence of the EBCASC parameter. If the EBCASC parameter is absent, the mnote 'EBCDIC TO ASCII-8 TRANSLATE TABLE NOT GENERATED' is printed. Otherwise, the EBCDIC to ASCII-8 translate table is generated. The macro then checks for the absence of the ASCEBC parameter. If this parameter is absent, the mnote 'ASCII-8 TO EBCDIC TRANSLATE TABLE NOT GENERATED' is printed. Otherwise, the ASCII-8 to ABCDIC translate table is generated.

The translate tables which are generated are as follows in Fig. 5

5.3 TMSCLOSE Macro

The TMSCLOSE macro first checks for the existence of its single parameter. It then tests for both a leading left and trailing right parenthesis. If it finds these, a register designator is assumed. If this register designator is some standard representation of register 1, the macro proceeds directly to the generation of the linkage code at sequence symbol ".LINK", since the register specified is the register in which the parameters are to be passed. If a register other than register 1 is specified, the macro generates an LR instruction to bring the contents of that register into register 1. If the operand is not a register specification it is assumed to be the symbolic address of a fullword in storage containing the DCB address. An L instruction is generated to bring this address into register 1. Finally, the TMSLINK macro is used

FIG. 5
TRANSLATE TABLES

```
      * EBCDIC TO ASCII-8 TRANSLATE TABLE
          0 1 2 3 4 5 6 7 8 9 A B C D E F
DC    X'000102030400000008000000000000000' 0
DC    X'00111200000000001819000000 1D0000' 1
DC    X'00000000000000000000000000000000' 2
DC    X'00000000000000000000000000000000' 3
DC    X'40000000000000000000094E5C484B0B' 4
DC    X'46000000000000000000041444A495B0D' 5
DC    X'4D4F000000000000000001C4C451B5E5F' 6
DC    X'43000000000000000005A0CA0475D42' 7
DC    X'00000000000000000000000000000000' 8
DC    X'00000000000000000000000000000000' 9
DC    X'00000000000000000000000000000000' A
DC    X'E1E2E3E4E5E6E7E8E9EAEBBBBCBDBEBF' B
DC    X'00A1A2A3A4A5A6A7A8A9000000000000' C
DC    X'00AAABACADAEAFB0B1B2000000000000' D
DC    X'0000B3B4B5B6B7B8B9BA000000000000' E
DC    X'505152535455565758590000000000000' F


      * ASCII-8 TO EBCDIC TRANSLATE TABLE
        0 1 2 3 4 5 6 7 8 9 A B C D E F
DC    X'000102030400000084A004F7B5F0000' 0
DC    X'00111200000000001819006 06A1D0000' 1
DC    X'00000000000000000000000000000000' 2
DC    X'00000000000000000000000000000000' 3
DC    X'405A7F7C5B6C507D4D5D5C4F6B604B61' 4
DC    X'F0F1F2F3F4F5F6F7F8F97A5E4C7$6E6F' 5
DC    X'00000000000000000000000000000000' 6
DC    X'00000000000000000000000000000000' 7
DC    X'00000000000000000000000000000000' 8
DC    X'00000000000000000000000000000000' 9
DC    X'7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6' A
DC    X'D7D8D9E2E3E4E5E6E7E8E9BBBCBDBEBF' B
DC    X'00000000000000000000000000000000' C
DC    X'00000000000000000000000000000000' D
DC    X'00B0B1B2B3B4B5B6B7B8B9BA00000000' E
DC    X'00000000000000000000000000000000' F
```

to generate the code that finds the entry point into the monitor for the close routine. The final instruction generated is a BALR RR, RC.

5.4 TMSCSIO Macro

The TMSCSIO macro employs the global binary variable &TMSRFB to find whether or not the problem programmer is maintaining the FB pointer in register 11 (R9).

The principal work within this macro is the analyzing of the keyword parameter OP and the generation of the bit settings in the options byte. After initialization, the presence or absence of the LENGTH operand is determined, and the corresponding bit is set in the local arithmetic variable &OPCODE. The first OP sub-parameter then is tested for one of the four allowable operations (READ, WRITE, CLEAR, or REWRITE), and the corresponding bit is set in &OPCODE. After this, a loop is entered to scan the remaining sub-parameters. As each one is recognized, a corresponding bit is set in &OPCODE, and special flag bits are set to prevent the acceptance of a duplicate parameter. When all sub-parameters have been exhausted, the code generation portion of the macro is entered at sequence symbol ".GENER".

If the principal operation is a WRITE, the positional operand representing the message is analyzed. If this operand consists of a string of characters delimited by apostrophes, the entire string is assembled in-line as the message text, preceded by a halfword message character count. A BAL instruction is generated both to skip over the message and to put its address into register (RP1). If the message parameter is determined to be a register designator, an LR instruction is generated to move the address into register 1. If the operand is a symbolic address, an LA instruction is generated to load the message address into register 1. An improper or omitted message specification for a WRITE operation causes a zero length message to be supplied along with two level A error statements. The code is then generated to locate the FB (if necessary), to clear the first byte of FBECB to binary zeros, and to move a one-byte opcode into FBRWOP. For a CLEAR operation, register 0 (RP0) is cleared to zero. For any other operation, the LENGTH parameter, if present, is tested to see whether it is a register specification or a symbolic address, and either an LR or LA instruction is generated to load the length into register 0 (RP0).

The TMSLINK macro is invoked to generate the code that finds the entry point in the monitor of the console input/output routine following this. If the user has specified both WAIT=DEFER and a parameter for the keyword operand RET, code is generated both to load the return address into register 14 (RR) and to branch to the monitor via register 15 (RC). For all other cases a standard BALR

instruction is generated. Finally, if WAIT=DEFER is not specified, the macro TMSWAIT is invoked with the proper OP and RET parameters.

5.5 TMSFREEM Macro

The TMSFREEM macro tests to see that its parameter both exists and is a register designator. If both of these tests are passed, the code is generated to clear register 0 (RP0) and to load register 1 (RP1) from the register specified. The TMSLINK macro is then invoked to generate code that finds the entry point of the GETMAIN/FREEMAIN routine, and the usual BALR instruction is generated to branch to it.

5.6 TMSGETM Macro

The TMSGETM macro first verifies the presence of its single parameter and then determines whether it is a register designator or a symbolic expression. A suitable LR or LA instruction is generated as needed. The macro TMSLINK is then invoked to generate code that finds the entry point in the monitor of the GETMAIN/FREEMAIN routine, and the usual BALR instruction is generated to branch to it.

5.7 TMSLINK Macro

The TMSLINK macro employs the global binary variable &TMSRFB to determine if the user program is maintaining the FB pointer in register 11 (R9). It also employs the global character variable &TMSRCR to indicate which register (if any) is being maintained by the user program as a pointer to the CR.

The only parameter for TMSLINK is a name which corresponds to one of the names defined in the Communication Region DSECT. If the global character variable &TMSRCR indicates that a pointer to the CR is being maintained, code is generated merely to load the required entry point address into register 15 (RC) from the CR. The macro is then exited. If the CR pointer is not being maintained, the global binary variable &TMSRFB is tested to see if the FB pointer is being maintained in register 11. If it is, code is generated both to obtain the CR pointer from the FB and to place it in register 15 (RC), assuming that a COPY FB statement has been encountered earlier in the assembly. Code is then generated to establish addressability to the CR via a USING statement, to load register 15 (RC) with the entry point address, and then to cancel the effect of the USING statement with a DROP statement.

In the event that neither the FB pointer nor the CR pointer is being maintained by the user program, code is generated to load the address of the FB into register 15 (RC) from the first word of the area pointed to by register 13 (RS). Addressability to the

FB is then established followed by code to load the CR address into register 15, followed again by code to load the entry point address into register 15. Thus, regardless of which registers are being maintained by the user program, code is generated to insure that the requisite entry point address is left in register 15 by the time this macro exits.

5.8 TMSOPEN Macro

The TMSOPEN macro begins extensive checking of its operands first by checking that the DSNAME operand exists and is less than or equal to eight characters in length. It then sets various flags based upon the RETURN, QUALIFY, and BUFFERS operands. At the same time a special output flag is set to indicate to the remainder of the macro whether WRITE operations will be acceptable for this data set. The FOR operand is then analyzed and four single-bit flags are set in patterns corresponding to their use in the standard O/S OPEN macro. The options recognized are INPUT, OUTPUT, UPDAT, INOUT, and OUTIN.

The DSORG operand is then tested in a manner similar to its analysis in expanding the O/S DCB macro instruction. It must both exist and be two or three characters in length. The first two characters are isolated and tested to see if they are either one of the four allowable TMS combinations or one of four further combinations, one of several single bit flags is set. These single bit flags will be used later to assemble the halfword PDSORG field in the expanded parameter list produced by the macro.

As with DSORG, the MACRF operand has a set of permissable values which itself is a sub-set of values permissable in the O/S DCB macro instruction. The analysis of this operand also is based upon the methodology used in the DCB macro. Since the MACRF operand may have several sub-parameters, an outer analysis loop is set up to analyze each sub-parameter at a time. The first character of each sub-parameter is isolated within this loop; this initial letter may have one of six values, only five of which are valid in TMS. If the initial letter is an E for EXCP, 20 bit flags are immediately set to a predetermined combination, and the MACRF operand analysis is finished. If the initial letter is P for PUT, the loop is exited without setting any flags for this particular sub-operand. If the initial character is R for READ or W for WRITE, the analysis continues by setting up an inner loop to analyze these remaining characters in the sub-operand considered to be qualifiers of the initial character. Each qualifier is isolated in turn and analyzed to see if it is one of the valid letters. If it is a valid letter, the settings of relevant DSORG flags are checked to see if the combination of the qualifier and the data set organization represents a valid specification. If it is valid, the proper bit flags are set, and the inner loop is repeated until all qualifiers have been exhausted. At this time the outer loop is repeated if any sub-operands remain.

If analysis has been completed on all MACRF sub-operands, a particular bit pattern has been established in 20 bit flags.

The third operand with a direct analog in the O/S DCB macro is the OPTCD operand. As before, the analysis of this operand is a sub-set of the analysis used in the DCB macro. Each character of the operand is analyzed to see if it is one of several acceptable letters. If an acceptable letter if found, the bit flags set for DSORG are checked to see that the combination of the letter and the data set organization is a valid specification. For each valid combination, one of eight bit flags is set.

The code generation portion for the macro begins, if necessary, with a BAL instruction branching around the in-line parameter list and loading the address of the parameter list into register 1 (RP1). The nine DSORG bit flags, followed by three binary zeros, followed by the four open option flags are assembled into a halfword binary bit pattern, and the corresponding DC instruction is generated. Following this, another halfword binary bit pattern, consisting of the first 16 MACRF flags, is assembled, and a second DC is generated. The eight OPTCD binary bit flags next are assembled into a single byte DC. The DC for an A-type address constant of length three containing the address specified as the SYNAD operand follows. At this point in the generation, if EXCP processing with appendages has been specified, the six DC instructions for character constants, each of length two, are assembled to represent the various appendage specifications. Following either of these appendage parameters, or the EODAD address if the appendage specifications are not present, comes the DC for a binary halfword count of the number of characters in the DSNAME operand. This is followed by the DC for a variable length field containing the EBDCIC character representation of the DSNAME operand. At this point in the generation, a direct branch is taken to the linkage generation portion of the macro, which is sequence symbol ".LINK".

Immediately following the above code comes the analysis of the second operand of the MF operand when it is specified for an execute form macro instruction. Verified as present, this second sub-operand is tested to see whether it is a register designator or a symbolic address. For a register designator, the proper LR instruction is generated to bring the parameter list address into register 1 (RP1); for a symbolic address an LA instruction is generated for the same purpose. The final portion of the generation is the linkage to the monitor system OPEN routine. The macro TMSLINK is invoked to generate the code that will place the entry point of the TMSOPEN routine in register 15 (RC). The final instruction generated for standard or execute form macro expansions is the usual BALR instruction.

5.9 TMSRETN Macro

The TMSRETN macro employs a global character variable &TMSRMSA

to determine whether the save area produced by the corresponding TMSSAVE macro was local, remote, or not supplied at all.

If the global variable indicates that the save area obtained was a remote save area, code is generated to place the current save address into register 1 (RP1), obtain the address of the previous save area from the second fullword of the current save area, put it in register 13 (RS), and invoke the TMSFREEM macro to free the core obtained for the remote save area. Control then passes to the common coding at the sequence symbol ".RESTORE". If it is determined that the save area involved is a local save area, the only unique code generated is that to obtain the address of the previous save area. If no save area at all is generated, control immediately passes to the common coding. The common coding generates code to zero the downward save area pointer in the third fullword of the previous save area, restore registers 14 (RR) through 12 (RB) from the previous save area, and return to the calling program via register 14 (RR).

5.10 TMSSAVE Macro

The TMSSAVE macro sets the global binary variable &TMSRFB to indicate to future macro instructions whether or not the problem programmer will maintain register 11 (R9) as a pointer to the FB. It sets the global character variable &TMSRMSA to show whether the save area obtained is a local save area, a remote save area, or non-existant. It sets the global character variable &TMSRCR to indicate which register, if any, the problem programmer guarantees to point to the CR. All three of these global variables play an important part in the expansion of most of the remaining macro instructions in the program.

The first operation performed by the expansion of TMSSAVE is a test on the parameter of the SAINCR keyword operand to see if it exceeds 4023 bytes. The purpose of this test is to issue a warning message if the increment plus the 72 byte save area exceeds 4095 bytes, thus indicating a possible requirement for an additional base register or registers. Following this test, code is generated to define the CSECT, to save the registers in the save area pointed to by register 13 (RS) by means of a STM instruction, to establish the permanent base register by means of an IR instruction, and to issue the USING instruction for the main base register. If RFB=NONE is not coded, the global binary variable &TMSRFB is set to indicate that the FB pointer will be maintained. The USING instruction for the FB pointer then is generated. If a register has been specified to act as the CR pointer, the specification is stored in the global character variable &TMSRCR and, if necessary, an LR instruction to load the CR pointer register from register 10 (R8) is issued. A USING instruction for the CR pointer follows the LR instruction.

The remainder of the macro obtains a save area if one is

required. The global character variable &TMSRMSA is set to the letter "N" as a default, and if SA=NONE has been coded, the macro is exited at this point. If SA=REMOTE, the default option, has been coded, &TMSRMSA is set to the letter "R", and one of two sequences of code is generated. If the sum of 72 and the value specified for SAINCR is less than or equal to 4095, the TMSGETM macro instruction is invoked with LV specified as a symbolic expression. However, when the amount of core to be obtained is greater than 4095 bytes, a halfword constant is generated in-line along with code to branch around this constant, load it into register 0 (RP0), and invoke the macro instruction TMSGETM LV=(RP0). Regardless of which form of code has been generated to obtain the remote save area, code is now generated both to clear the first 72 bytes of the save area and to load the pointer to the new save area into the register specified by the keyword operand RWORK. The macro then branches to the sequence symbol ".CHAINSA". If SA=LOCAL has been specified, &TMSRMSA is set to the letter "L" and an 18-fullword save area is generated in-line with code both to branch around it and put its address into the register specified in the keyword operand RWORK. For both local and remote save areas, code is then generated to move the FB pointer from the first word of the old save area to the first word of the new save area and to link the two save areas in standard O/S fashion, using the register specified by the keyword operand RWORK as a work register.

5.11 TMSWAIT Macro

The TMSWAIT macro uses the global binary variable &TMSRFB to determine whether or not the problem programmer is maintaining register 11 (R9) as a pointer to the FB.

The macro first tests for the special case where ECB=FBECB has been coded and the FB pointer is not being maintained by the problem programmer. If both these conditions exist, code is generated to obtain the FB pointer from the first word of the current save area and establish addressability to the FB by means of a USING instruction. An LA instruction to load register 1 (RP1) with the ECB address is then generated, followed by a DROP instruction to terminate FB addressability. If this special case does not exist, the only instruction generated is the LA instruction to load register 1 with the ECB address.

The TMSLINK macro is invoked to generate the necessary code to locate the WAIT routine entry point in the monitor. If a return address has been supplied by use of the RET keyword operand, and OP=READ has not been coded, an LA instruction is generated to load register 14 (RR) with the return address, followed by a BR instruction to branch to the WAIT routine's entry point. In all other cases, the usual BALR instruction is generated. If OP=READ has not been coded, the macro is exited at this point. If it has, an SR instruction is generated to clear register 1. If the problem programmer is maintaining the FB pointer, code is generated both

to calculate the start of text and length of text in the buffer and to leave these in the proper registers. This code consists of an IC instruction to place the text offset from FBBUFOFF into register 1; an L instruction followed by an N instruction to obtain the buffer address from FBBUFPTR and leave it in register 0; an AR instruction to add the buffer address in register 0 to the offset in register 1, leaving the resulting pointer to the start of text in register 1; and an LH instruction to put the length of text from FBLMLNTH into register 0. If the problem programmer is not maintaining the FB pointer in register 11, the same code is generated first, preceded by both an L instruction and a USING instruction to obtain the FB address and to establish addressability, and then followed by a DROP instruction to end addressability.

APPENDICES

APPENDIX 1: TMS MODULE NAMES AND ENTRY POINTS

MODULE	ENTRY POINTS
TMSHSKP	TMSHSKP
TMSBLOCK	TMSBLOCK TMSBLGTH TMSLSTFB
TMSPJOB	TMSPJOB
TMSPLOAD	TMSPLOAD
TMSBEGIN	TMSBEGIN TM3CRADR TMSSYSSV
TMSCNSL	TMSCNSL TMSREADY
TMSCSIO	TMSCSIO TMSWRDEQ TMSRDRST TMSCSIOR
TMSWAIT	TMSWAIT TMSDWAIT TMSQWAIT
TMSTREND	TMSTREND TMSCHEND
TMSGMFM	TMSGMFM
TMSOPEN	TMSOPEN
TMSCLOSE	TMSCLOSE TMSPCLOS
TMSPURGE	TMSPURGE
TMSGTSLE	TMSGTSLE
TMSDEBUG	TMSDEBUG

APPENDIX 2A: COMMUNICATION REGION--'CR'

```
OFFSET
          *
          *          The general communication region is located in the
          *          resident monitor code.  A pointer to it is located
          *          in each function block as 'FBCR'.
          *
          *          Aligned on Fullword Boundary
          *
000000    CR         Dsect
          *
000000    CRWAIT     DS     A           Address of EP 'TMSWAIT'
000004    CRCSIO     DS     A           Address of EP 'TMSCSIO'
000008    CRPURGE    DS     A           Address of EP 'TMSPURGE'
00000C    CRLOAD     DS     A           Address of EP 'TMSPLOAD'
000010    CRGMFM     DS     A           Address of EP 'TMSGMFM'
000014    CRSNAP     DS     A           Address of EP 'TMSSNAP'
000018    CROPEN     DS     A           Address of EP 'TMSOPEN'
00001C    CRCLOSE    DS     A           Address of EP 'TMSCLOSE'
000020    CRWL       DS     A           Pointer to WAIT List
000024    CRWLLAST   DS     A           Pointer to Current Last Entry
000028    CRECB      DS     A           CR Dummy ECB
          *
          CRWRITE    EQU    X'20'       Line Newly Available for WRITE
          CRERPOLL   EQU    X'10'       Line Available for Poll RESTART
          *
00002C    CRIND      DS     XL1         CR Indicators
          *
          CRCEERR    EQU    X'80'       Error in Channel End Routine
          *
00002D    CRABCODE   DS     XL1         Abnormal End Code
00002E    CRWLX      DS     H           Offset to WAIT List Extension
000030    CRLIBDCE   DS     A           Address of Program Library DCB
000034    CRSNPDCB   DS     A           Address of Snap DCB
000038    CRLOGDCB   DS     A           Address of System Log DCB
00003C    CRFBCHN    DS     A           Start of FB Chain
000040    CRQFLAGS   DS     0XL1        Current Needs of Queued FB's
          *
          CRQEND     EQU    X'80'       No FB's Queued at This Time
          CRQWRITE   EQU    X'40'       1 or More Queued for WRITE
          CRQRPOLL   EQU    X'20'       1 or More Queued for Poll RESTART
          CRQSNAP    EQU    X'10'       1 or More Queued for SNAPSHOT
          CRQPLOAD   EQU    X'08'       1 or More Queued for Program LOAD
          *
000040    CRQUEUE    DS     A           PTR to Queue of FB's
000044    CRPICA     DS     F           Save Area for PICA
000048    CRGISLE    DS     A           Address of EP 'TMSGTSLE'
000050    CREND      DS     0D          End of Communication Region
          CRLENGTH   EQU    CREND-CR    Length of Communication Region
                     COPY   FB
```

CF FB TDECB

GENERAL COMMUNICATION REGION--'CR'

0 (0)	CRWAIT Address of EP 'TMSWAIT'	
4 (4)	CRCSIO Address of EP 'TMSCSIO'	
8 (8)	CRPURGE Address of EP 'TMSPURGE'	
12(C)	CRPLOAD Address of EP 'TMSPLOAD'	
16(10)	CRGMFM Address of EP 'TMSGMFM'	
20(14)	CRSNAP Address of EP 'TMSSNAP'	
24(18)	CROPEN Address of EP 'TMSOPEN'	
28(1C)	CRCLOSE Address of EP 'TMSCLOSE'	
32(20)	CRWL Address of WAIT List	
36(24)	CRWLLAST Address of Last Entry	
40(28)	CRECB Communication Region Event Control Block	
44(2C) CRIND CR Indicators	45 CRABCODE Abnormal End Code	46(2E) CRWLX Offset to WAIT List Extension
48(30)	CRLIBDCB Address of Program Library DCB	
52(34)	CRSNPDCB Address of Snap DCB	
56(38)	CRLOGDCB Address of System Log DCB	
60(3C)	CRFBCHN Start of FB Chain	
64(40) CRQFLAGS Queued FB Needs		65(41) CRQUEUE PTR to Queue of FB's
68(44)	CRPIE Save Area for PIE	
72(48)	CRGTSLE Address of EP 'TMSGTSLE'	

GENERAL COMMUNICATIONS REGION--'CR'

OFFSET	BYTES & ALIGNMENT	FIELD NAME	HEX. DIG.	FIELD DESCRIPTION, CONTENTS, MEANING
0 (0)	4	CRWAIT		Address of Entry Point for TMSWAIT Module
4 (4)	4	CRCSIO		Address of Entry Point for TMSCSIO Module
8 (8)	4	CRPURGE		Address of Entry Point for TMSPURGE Module
12 (C)	4	CRPLOAD		Address of Entry Point for TMSPLOAD Module
16 (10)	4	CRGMFM		Address of Entry Point for TMSGMFM Module
20 (14)	4	CRSNAP		Address of Entry Point for TMSSNAP Module (not used at this time)
24 (18)	4	CROPEN		Address of Entry Point for TMSOPEN Module
28 (1C)	4	CRCLOSE		Address of Entry Point for TMSCLOSE Module
32 (20)	4	CRWL		Address of the WAIT List
36 (24)	4	CRWLLAST		Address of the Last Current Entry on WAIT List
40 (28)	4	CRECB		Communication Region Dummy Event Control Block
		xx.. xxxx		(Reserved Bits)
		..1.		Line Newly Available for WRITE
		...1		Line Available for Polling RESTART
44 (2C)	1	CRIND		CR Indicators
		.xxx xxxx		(Reserved Bits)
		1...		Error in Channel End Routine
45 (2D)	. 1	CRABCODE		Abnormal End Code
			28	Unsuccessful Polling Halt
			29	Buffer Unavailable for WRITE Initialization
			2A	Start of WRITE Unsuccessful (CSIO)
			2B	Buffer Unavailable for Input
			2C	Error in Channel was not Timeout, Lost Data, or Data Check
			2D	RESTART Parameter Set Already Exists (CSIO)
			32	Invalid Terminal List Offset
			33	Both READ and WRITE Operations Specified
			34	Neither READ or WRITE Operations Specified
			35	Invalid Buffer Flags at READ Completion
			37	Active Polling Count Invalid at READ Completion

GENERAL COMMUNICATIONS REGION -- 'CR'

OFFSET	BYTES & ALIGNMENT	FIELD NAME	HEX. DIG.	FIELD DESCRIPTION, CONTENTS, MEANING
			38	Improper Flags in DECUFLGS Field of TDECB
			39	Improper Buffer Flags at End of WRITE
			3A	Channel Error Processing for CE, DE, UC Redundant
			3B	Error is not Channel End, Device End, Unit Check
			3C	RESTART Parameter Set Already Exists (TREND)
			3D	Start of WRITE Unsuccessful (TREND)
			3E	Start of READ Unsuccessful (TREND)
			FF	I/O Error Recovery Failure
46 (2E)	2	CRWLX		Offset to WAIT List Extension
48 (30)	4	CRLIBDCB		Address of Program Library DCB
52 (34)	4	CRSNPDCB		Address of Snap DCB
56 (38)	4	CRLOGDCB		Address of System Log DCB
60 (3C)	4	CRFBCHN		Start of FB Chain
64 (40)	1	CRQFLAGS		Current
		xxx		(Reserved Bits)
		1...		No. FB's Queued at This Time
		.1..		1 or More Queued for WRITE
		..1.		1 or More Queued for Polling RESTART
		...1		1 or More Queued for Snapshot
		.... 1...		1 or More Queued for Program Load
65 (41)	. 1	CRQUEUE		Address of Queue of FB's
68 (44)	4	CRPIE		Save Area for PIE
72 (48)	4	CRGTSLE		Address of Entry Point for TMSGTSLE Module

APPENDIX 2B: FUNCTION BLOCK--'FB'

```
OFFSET  *
        *       The function control block exists for each operating
        *       terminal and contains control information, work areas,
        *       etc.
        *       Aligned on Fullword Boundary
        *
000000  FB      DSECT
        *
000000  FBSAVE   DS    16F       Special Function Save Area
000040  FBECB    DS    F         Event Control Block
000044  FBFDLAGS DS    0XL1      FB Flags
        *
        FBXMIERR EQU   X'80'     Transmission Error
        FBDSCNCT EQU   X'40'     Disconnect
        FBDEBUG  EQU   X'20'     Debugging on Terminal
        *
000044  FBCB     DS    A         Address of Communications Region
00004B  FBEFLAG  DS    0XL1
        *
        FBEPJOB  EQU   X'01'     Last Entry Thru PJOB
        FBEPJOBP EQU   X'02'     Last Entry Thru Purged in PJOB
        FBEPLOAD EQU   X'03'     Last Entry Thru PLOAD
        FBECNSL  EQU   X'04'     Last Entry Thru CNSL
        FBEREADY EQU   X'05'     Last Entry Thru READY
        FBECSIO  EQU   X'06'     Last Entry Thru CSIO
        FBEWRDEQ EQU   X'07'     Last Entry Thru WRDEQ
        FBERDRST EQU   X'08'     Last Entry Thru RDRST
        FBECSIOR EQU   X'09'     Last Entry Thru CSIOR
        FBEWAIT  EQU   X'10'     Last Entry Thru WAIT
        FBEDWAIT EQU   X'11'     Last Entry Thru DWAIT
        FBEQWAIT EQU   X'12'     Last Entry Thru QWAIT
        FBETREND EQU   X'13'     Last Entry Thru TREND
        FBEGMFM  EQU   X'14'     Last Entry Thru GMFM
        FBEOPEN  EQU   X'15'     Last Entry Thru OPEN
        FBECLOSE EQU   X'16'     Last Entry Thru CLOSE
        FBEPCLOS EQU   X'17'     Last Entry Thru PCLOS
        FBEPURGE EQU   X'18'     Last Entry Thru PURGE
        FBEDEBUG EQU   X'19'     Last Entry Thru DEBUG
        *
000048  FBCHAIN  DS    A         Pointer to Next FB
00004C  FBRLN    DS    0XL1      Relative Line No. for This Line
00004C  FBTCHAIN DS    A         Next FB Chained for This Line
000050  FBQFLAGS DS    0XL1      Reason for Which FB is Queued
        *
        FBQEND   EQU   X'80'     Last FB on Queue
        FBQWRITE EQU   X'40'     FB Queued for WRITE to Terminal
        FBQRPOLL EQU   X'20'     FB Queued for READ Poll RESTART
        FBQSNAP  EQU   X'10'     FB Queued for SNAPSHOT Routine
        FBQPLOAD EQU   X'08'     FB Queued for New Program LOAD
        *
000050  FBQUEUE  DS    A         PTR to Next Queued FB
```

APPENDIX 2B (CONT.)

OFFSET				
000054	FBBLKCHN	DS	A	Start of User Storage Block Chain
000058	FBDCBCHN	DS	A	Start of User DCB Chain
00005C	FBTLOFF	DS	H	Terminal List Offset
00005E	FBPOLL	DS	H	Polling Characters for Console
000060	FBRWOP	DS	0XL1	READ/WRITE Opcode for Terminal
	*			
	FBRWRITE	EQU	X'80'	Bit to Indicate WRITE
	FBRWPE	EQU	X'40'	Bit to Indicate PRE-ERASE
	FBRWCRAW	EQU	X'20'	Bit to Indicate CR After WRITE
	FBRWEDIT	EQU	X'08'	Bit to Indicate Edit Before WRITE
	FBRWNLEW	EQU	X'04'	Bit to Indicate NL Before WRITE
	FBRWLREG	EQU	X'02'	Bit to Indicate Length in RPO
	FBRWRWRT	EQU	X'01'	Bit to Indicate REWRITE
	*			
000060	FBDECB	DS	A	DECB for Line Associated With FB
000064	FBBUFOFF	DS	0XL1	Offset to Text in Buffer
000064	FBBUFPTR	DS	A	PTR to Buffer Attached to FB
000068	FBWFLAGS	DS	0XL1	Working Flags
	*			
	FBWF	EQU	X'80'	
000068	FBWRKPTR	DS	A	Working Pointer
00006C	FBLCOUNT	DS	H	Current Line Count
00006E	FBCLLGTH	DS	H	Length of Current Line
000070	FBLMLNTH	DS	H	Length of Previous Message
000072	FBTERMNO	DS	CL2	Terminal Number in EBCDIC
000074	FBNAME	DS	CL4	Name of Current User
000078	FBPNAME	DS	CL8	Name of Current User Program
000080	FBEND	DS	0F	End of Function Block
	FBLENGTH	DS	0XL (FBEND-FBSAVE)	Length of Function Block

FUNCTION CONTROL BLOCK--'FB'

0 (0)	FBSAVE Save Area for Users Register		
60 (3C)			
64 (40)	FBECB Event Control Block		
68 (44)	FBFLAGS FB Flags	FBCR Address of CR	
72 (48)	FBCHAIN Pointer to Next FB		
76 (4C)	FBRLN Relative Line No.	FBTCHAIN Next FB Chained for Their Line	
80 (50)	FBQFLAGS Reason FB is Queued	FBQUEUE Pointer to Next Queued FB	
84 (54)	FBBLKCHN Start of User Storage Block Chain		
88 (58)	FBDCBCHN Start of User DCB Chain		
92 (5C)	FBTLOFF Terminal List Offset	94 (5E)	FB Poll Polling Characters
96 (60)	FBRWOP READ/WRITE Opcode	FBDECB DECB for Line Associated with This FB	
100 (64)	FBBUFOFF Offset to Text	FBBUFPTR Address of Buffer Attached to This FB	
104 (68)	FBWFLAGS Working Flags	FBWRUPTR Working Pointer	
108 (6C)	FBBLCOUNT Current Line Count	110 (6E)	FBCLLGTH Length of Current Line
112 (70)	FBLMLNTH Length of Previous Message	114 (72)	FBTERMNO Terminal Number
116 (74)	FBNAME Name of Current User		
120 (78)	FBPNAME Name of Current User Program		

FUNCTION CONTROL BLOCK--'FB'

OFFSET	BYTES AND ALIGNMENT	FIELD NAME	FIELD DESCRIPTION, CONTENTS, MEANING
0 (0)	64	FBSAVE	Save Area for TMS Functions
64 (40)	4	FBECB	Event Control Block
68 (44)	1	FBFLAGS	Function Block Flags:
		 xxxx	(Reserved Bits)
		1...	FBXMTERR - Transmission Error
		.1..	FBDSCNCT - Disconnect
		..1.	FBDEBUG - Debugging of Terminal
68 (44)	4	FBCR	Address of Communications Region
72 (48)	4	FBCHAIN	Pointer to Next FB
76 (4C)	1	FBRLN	Relative Line Number for This Line
76 (4C)	4	FBTCHAIN	Next FB Chained for This Line
80 (50)	1	FBQFLAGS	Reason for Which FB is Queued:
		 xxxx	(Reserved Bits)
		1...	FBQUEND - Last FB on Queue
		.1..	FBQWRITE - FB Queued for WRITE to Terminal
		..1.	FBQRPOLL - FB Queued for READ Polling Restart
		...1	FBQSNAP - FB Queued for Snapshot
		.... 1...	FBQPLOAD - FB Queued for Program Load
80 (50)	4	FBQUEUE	Pointer to Next Queued FB
84 (54)	4	FBBLKCHN	Start of User Storage Block Chain
88 (58)	4	FBDCBCHN	Start of User DCB Chain
92 (5C)	2	FBTLOFF	Offset in Terminal List to Entry for This FB

FUNCTION CONTROL BLOCK--'FB'

OFFSET	BYTES & ALIGNMENT	FIELD NAME	FIELD DESCRIPTION, CONTENTS, MEANING
94 (5E)	. . 2	FBPOLL	Polling Characters for This FB
96 (60)	1	FBRWOP	READ/WRITE Opcode
96 (60)	4	FBDECB	DECB for the Line Associated With This FB
100 (64)	1	FBBUFOFF	Offset to Beginning of Text in Input Buffer
100 (64)	4	FBBUFPTR	Address of Buffer Attached to This FB
104 (68)	1	FBWFLAGS	Working Flags
104 (68)	4	FBWRKPTR	Working Pointer
108 (6C)	2	FBBLCOUNT	Current Line Count
110 (6E)	. . 2	FBCLLGTH	Length of Current Line
112 (70)	2	FBLMLNTH	Length of Previous Message
114 (72)	. . 2	FBTERMNO	Terminal Number in EBCDIC
116 (74)	4	FBNAME	Name of Current User
120 (78)	8	FBPNAME	Name of Current User Program

APPENDIX 2C: TELEPROCESSING DATA EVENT CONTROL BLOCK (TDECB)

```
OFFSET
            *
            *           The data event control block for teleprocessing via
            *           BTAM consists of 40 bytes defined by IBM plus user-
            *           defined fields defined for the TMS system.
            *
            *           Aligned on Fullword Boundary
            *
000000      TDECB       DSECT
            *
000000      DECSDECB    DS      F       Standard Event Control
000004      DECTYPE     DS      H       Block Operation Type
            *
            *           Standard BTAM Optype Codes (Seccnd Byte)
            *
            DECRTI      EQU     X'01'   READ Initial
            DECWTI      EQU     X'02'   WRITE Initial
            DECWTIR     EQU     X'82'   WRITE Initial With Reset
            DECRTT      EQU     X'03'   READ Continue
            DECRTP      EQU     X'07'   READ Repeat
            DECWTA      EQU     X'08'   WRITE POSITIVE ACKNOWLEDGE
            DECWTSR     EQU     X'8E'   WRITE Erase With Reset
            *
000006      DECLNGTH    DS      H       Area Length
000008      DECONLTT    DS      0CL1    Reserved For On-Line Terminal Test
000008      DECDCBAD    DS      A       Address of DCB
00000C      DECAREA     DS      A       Address of Area
000010      DECSENSO    DS      C       1st Sense Byte
            *
            DECSCMRJ    EQU     X'80'   Command Reject
            DECSINTV    EQU     X'40'   Intervention Required
            DECSBOCK    EQU     X'20'   Busout Parity Check
            DECSEQCK    EQU     X'10'   Equipment Check
            DECSDTCK    EQU     X'08'   Data Check
            DECSOVRN    EQU     X'04'   Overrun
            DECSLOST    EQU     X'02'   Lost Data
            DECSTOUT    EQU     X'01'   Timeout
            *
000011      DECSENS1    DS      C       2nd Sense Byte
000012      DECCOUNT    DS      H       Residual Count
000014      DECCMCOD    DS      0CL1    Command Code
000014      DECENTRY    DS      A       Address of Terminal List
000018      DECFLAGS    DS      C       Status Flags
            *
            DECFNEGR    EQU     X'04'   Negative Response to Polling
            *
000019      DECRLN      DS      C       Relative Line Number
00001A      DECRESPN    DS      H       Response Fields
00001C      DECTPCOD    DS      H       Teleprocessing Opcode
00001D      DECERRST    DS      C       Error Status
00001E      DECCSWST    DS      C       CSW Status
            *
```

APPENDIX 2C: TDECB (CONT.)

OFFSET				
	DECCSWCE	EQU	X'08'	Status Flag--Channel End
	DECCSWDE	EQU	X'04'	Status Flag--Device End
	DECCSWUC	EQU	X'02'	Status Flag--Unit Check
	DECCSWUE	EQU	X'01'	Status Flag--Unit Exception
	DECCSWIL	EQU	X'40'	Status Flag--Incorrect Length
	*			
000020	DECADRPT	DS	A	Address of Current Addressing Entry
000024	DECPOLPT	DS	A	Address of Current Polling Entry
	*			
	*	Fields Specific to TMS		
	*			
000028	DECFBCHN	DS	A	PTR to Chain of FB's for Line
00002C	DECAPCNT	DS	H	Active Polling Count
00002E	DECWWCNT	DS	H	Waiting-to-Write Count
000030	DECTTYPE	DS	0XL1	Terminal Type
	*			
	DECCRT	EQU	X'80'	Bit to Indicate CRT Display
	DECTYPEW	EQU	X'40'	Bit to Indicate Typewriter
	DECMULTI	EQU	X'20'	Bit to Indicate Shared Line
	DECSEPLF	EQU	X'10'	Separate Line Feed Required
	*			
000030	DECTTIN	DS	A	Address of Inbound Translate Table
000034	DECUFLGS	DS	0XL1	TMS Flags
	*			
	DECUFRIP	EQU	X'80'	READ Polling in Progress
	DECUFWIP	EQU	X'40'	Writing in Progress
	DECUFPIN	EQU	X'20'	Polling Interrupted to Write
	DECUFWTW	EQU	X'10'	Another Terminal Waiting to Write
	DECUFPRS	EQU	X'08'	Polling RESET in Progress
	DECUFACK	EQU	X'04'	POSITIVE ACKNOWLEDGMENT Needed
	*			
000034	DECTTOUT	DS	A	Address of Outbound Translate Table
000038	DECCPNUL	DS	0XL1	Characters Per Null for CR
000038	DECTTLIN	DS	A	Address of Inbound L/U Translate Table
00003C	DECRSAVE	DS	CL11	Save Area for READ Parameters
000047	DECCPLIN	DS	XL1	Characters Positions Per Line
000048	DECMAXNL	DS	0XL1	Maximum Number of Lines on Screen
000048	DECTLIST	DS	A	Address of Terminal List

DATA EVENT CONTROL BLOCK FOR TELEPROCESSING--'TDECB'

0 (0)	DECSDECB Event Control Block		
4 (4)	DECTYPE Operation Type	6 (6)	DECLNGTH Area Length
8 (8)	DECONLTT (Reserved)	DECDCBAD Address of DCB	
12 (C)	DECAREA Address of Area		
16 (10)	DECSENSØ 1st Sense Byte	17 (11) DECSENS1 2nd Sense Byte	18 (12) DECCOUNT Residual Count
20 (14)	DECCMCOD Command Code	DECENTRY Address of Terminal List	
24 (18)	DECFLAGS Status Flags	25 (19) DECRLN Relative Line No.	26 (1A) DECRESPN Addressing Response Field
28 (1C)	DECTPCOD Operation	29 (1D) DECERRST I/O ERROR Status	30 (1E) DECCSWST CSW Status
32 (20)	DECADRPT Address of Current Addressing Entry		
36 (24)	DECPOLPT Address of Current Polling Entry		
40 (28)	DECFBCHN Pointer to Chain of FB		
44 (2C)	DECAPCNT Active Polling Count	46 (2E)	DECWWCNT Waiting to Write Count
48 (30)	DECTTYPE, DECTTIN Terminal Type, Address of Inbound Translate Table		
52 (34)	DECUFLGS, DECTTOUT TMS Flags, Address of Outbound Translate Table		
56 (38)	DECCPNUL, DECTTLIN Characters/Null for CR, Address of Inbound L/U Translate Table		
60 (3C)	DECRSAVE Save Area for READ Parameters		
		71 (47)	DECCPLIN Positions/Line
72 (48)	DECMAXNL, DECTLIST Maximum Number Lines/Screen, Address of Terminal List		

DATA EVENT CONTROL BLOCK FOR TELEPROCESSING--'TDECB'

OFFSET	BYTES & ALIGNMENT	FIELD NAME	HEX. DIG.	FIELD DESCRIPTION, CONTENTS, MEANING
0 (0)	4	DECSDECB		Event Control Block
4 (4)	2	DECTYPE		Operation Type
			01	DECRT1 - READ Initial
			02	DECWT1 - WRITE Initial
			82	DECWT1R - WRITE Initial with Reset
			03	DECRTT - READ Continue
			07	DECRTP - READ Repeat
			08	DECWTA - WRITE POSITIVE ACKNOWLEDGE
			8E	DECWTSR - WRITE Erase with Reset
6 (6)	. . 2	DECLNGTH		Length of Buffer or Message Area
8 (8)	1	DECONLTT		Reserved for On-Line Terminal Test
9 (9)	. 3	DECDCBAD		Address of Associated DCB
12 (C)	4	DECAREA		Address of Buffer or Message Area
16 (10)	1	DECSENSØ		First Sense Byte
		1...		DECSCMRJ - Command Reject
		.1..		DECSINTV - Intervention Required
		..1.		DECSBOK - Busout Parity Check
		...1		DECSEQCK - Equipment Check
		.... 1...		DECSDTCK - Data Check
		.... .1..		DECSOVRN - Overrun
		.... ..1.		DECSLOST - Last Data
		1		DECSTOUT - Timeout
17 (11)	. 1	DECSENS1		Second Sense Byte (Reserved)
18 (12)	. . 2	DECCOUNT		Residual Count From CSW for Last CCW Executed
20 (14)	1	DECCMCOD		Command for Which Error Occurred
21 (15)	. 3	DECENTRY		Address of the Terminal List
24 (18)	1	DECFLAGS		Status Flags
		xxxx x.xx		(Reserved Bits)
		.... .1..		DECFNEGR - Regative Response to Polling
25 (19)	. 1	DECRLN		Relative Line Number
26 (1A)	. . 2	DECRESPN		Addressing Response Field
28 (1C)	1	DECTPCOD		Teleprocessing Operation Code
29 (ID)	. 1	DECERRST		I/O ERROR Status Flags
		1...		SIO Resulted in a Condition Code of 3
		.1..		Undefined Error Condition
		..1.		I/O ERROR in Error Running Routine
		...x .xxx		(Reserved Bits)
		.... 1...		Disable Issued to a Switched-Connected Line

DATA EVENT CONTROL BLOCK FOR TELEPROCESSING--'TDECB'

OFFSET	BYTES & ALIGNMENT	FIELD NAME	HEX. DIG.	FIELD DESCRIPTION, CONTENTS, MEANING
30 (1E)	. . 2	DECCSWST		CSW Status
32 (20)	4	DECADRPT		Address of Current Addressing Entry
36 (24)	4	DECPOLPT		Address of Current Polling Entry
40 (28)	4	DECFBCHN		Pointer to Chain of FB's for Line
44 (2C)	2	DECAPCNT		Active Polling Count
46 (2E)	. . 2	DECWWCNT		Waiting to Write Count
48 (30)	1	DECTTYPE,		
		DECTTIN		Terminal Type
49 (31)	. 3	DECTTIN		Address of Inbound Translate Table
52 (34)	1	DECUFLGS,		
		DECTTOUT		TMS Flags
		1...		DECUFRIP - READ Polling in Progress
		.1..		DECUFWIP - Writing in Progress
		..1.		DECUFPIN - Polling Interrupted to Write
		...1 .. .		DECUFWTW - Another Terminal Waiting to Write
		.... 1...		DECUFPRS - Polling RESET in Progress
		.... .1..		DECUFACK - POSITIVE ACKNOWLEDGE Needed
		xx		(Reserved Bits)
53 (35)	. 3	DECTTOUT		Address of Outbound Translate Table
56 (38)	1	DECCPNUL,		
		DECTTLIN		Characters per Null for CR
57 (39)	. 3			Address of Inbound L/U Translate Table
60 (3C)	11	DECRSAVE		Save Area for READ Parameters
71 (47)	. . . 1	DECCPLIN		Character Position per Line
72 (48)	1	DECMAXNL,		
		DECTLIST		Maximum Number of Lines per Screen
73 (49)	. 3	DECTLIST		Address of Terminal List

APPENDIX 3: LOAD MODULE ELEMENTS

TMS is divided into two load modules, each of which is comprised of multiple object modules or control sections. The TMS load modules are made up as follows:

LOAD MODULES	INCLUDED OBJECT MODULES
TMSHSKP	TMSHSKP
	TMSBLOCK
TMSEXEC	TMSPJOB
	TMSPLOAD
	TMSBEGIN
	TMSCNSL
	TMSCSIO
	TMSWAIT
	TMSTREND
	TMSGMFM
	TMSOPEN
	TMSCLOSE
	TMSPURGE
	TMSDEBUG

APPENDIX 4: STANDARD LIST OF I/O MODULES

BSAM MODULES:

IGG0198A	BSAM READ/WRITE MODULE(384 BYTES)
IGG0198B	BSAM CHECK MODULE (96 BYTES - MODIFIED)

BDAM MODULES:

IGG019KA	BDAM FOUNDATION MODULE (1480 BYTES)
IGG019KC	BDAM RELATIVE TRACK CONVERSION MODULE (280 BYTES)
IGG019KE	BDAM RELATIVE BLOCK CONVERSION MODULE (304 BYTES)
IGG019KI	BDAM CHANNEL PGM FOR KEY SEARCH (152 BYTES)
IGG019KK	BDAM CHANNEL PGM FOR ID SEARCH (176 BYTES)
IGG019KM	BDAM WRITE ADD FORMAT U OR V (584 BYTES)
IGG019KO	BDAM WRITE ADD FORMAT F (264 BYTES)
IGG019KS	BDAM START I/O APPENDAGE (64 BYTES)
IGG019KU	BDAM CHANNEL END APPENDAGE (132 BYTES)
IGG019KW	BDAM KEY EXTENDED SEARCH (200 BYTES)
IGG019KY	BDAM SELF-FORMAT EXTENDED SEARCH (200 BYTES)
IGG019LA	BDAM PRE-FORMAT EXTENDED SEARCH (200 BYTES)
IGG019LC	BDAM END OF EXTENT APPENDAGE (168 BYTES)
IGG019LI	BDAM CHECK MODULE (240 BYTES)

BTAM MODULES:

IGG019MA	BTAM READ/WRITE MODULE (1568 BYTES)
IGG019MB	BTAM CE & AE APPENDAGES (2744 BYTES)
IGG019M3	BTAM SANDERS 720 DDM (312 BYTES - MODIFIED)